30 Hour BASIC

ZX81 Edition

CLIVE PRIGMORE

BBC
NEC

THE COMPUTER PROGRAMME BBC TV
NATIONAL EXTENSION COLLEGE

NATIONAL EXTENSION COLLEGE

30 Hour BASIC

ZX81 edition

by
Clive Prigmore
(Orpington College of Further Education)
adapted for ZX81 by Richard Freeman and Robert Horvath

NATIONAL EXTENSION COLLEGE CORRESPONDENCE TEXTS
COURSE NUMBER M27X

ISBN 0 86082 301 6

Designer: Peter Hall
Photographs: Michael Manni, Cambridge.
Cover photograph: Reeve photography, Cambridge.
Printed by NEC Print.
Cover printed by Burlington Press, Foxton, Cambridgeshire.
First printing 1982
Reprinted with corrections 1982

Contents

How to use this course		5
Unit 1	Simple statements and commands	7
Unit 2	Making decisions	35
Unit 3	Strings	63
Unit 4	Lists	89
Unit 5	An end to strings and PRINT	117
Unit 6	Mainly about dice and games	145
Unit 7	Handling numbers	169
Unit 8	An introduction to data processing	197
	Index	224

How to use this course

Aims of the course
Quite simply, to help you learn to use a microcomputer with confidence. To do that you need to master three things: (a) the BASIC language; (b) planning good program structures; and (c) using the keyboard. This course teaches you the first two. Your computer will teach you the third!

30 Hour BASIC isn't all there is to know about BASIC but it does cover all the essentials. Once you've completed the course, you will be ready to use a textbook on BASIC or for the second stage BASIC course which will shortly be available from NEC.

Do I need a microcomputer?
You can do the course whether or not you have a microcomputer. All you have to do is to choose one of the following ways of working through the course:

Self-instructional use: With a ZX81 microcomputer: do all the Exercises and self-assessment questions (SAQ's) and key in all the programs marked $\boxed{K}$. Without a microcomputer: do all the Exercises and SAQ's but omit the items marked $\boxed{K}$.

FlexiStudy use: With your own ZX81 microcomputer: do all the Exercises and SAQ's at home and key in at home all the programs marked $\boxed{K}$. Test your assignment answers on your own microcomputer. Then take/send your problems to your local FlexiStudy centre. Without your own microcomputer: do all the Exercises and SAQ's but ignore the items marked $\boxed{K}$. Then do the assignment questions and take these to your local FlexiStudy centre to run on their ZX81's.

NEC correspondence students: With your ownZX81 microcomputer: do all the Exercises and SAQ's at home and key in all the items marked $\boxed{K}$. Test your assignments on your own ZX81 before sending them to your NEC correspondence tutor. If the assignment programs don't run properly, tell your tutor what response you are getting from your microcomputer. Without your own microcomputer: do all the Exercises and SAQ's at home but ignore the items marked $\boxed{K}$. Do the assignment questions and post these to your NEC tutor.

Structure of the course
The course is in 8 Units (see Contents). Each Unit includes:

Examples: These are problems which we solve completely for you in the text.

Self Assessment Question (SAQ's): We ask you to stop and quickly check that you have understood a new idea that we have introduced. Answers to these always appear at the end of the Unit in which the SAQ occurs.

Exercises: These are longer problems for you to try. Answers appear at the end of the Unit in which the Exercise occurs.

$\boxed{K}$ which stands for key. This is where we think you could find it helpful to key a

program into your own ZX81.

Assignment: These are questions for you to answer and send to your tutor for marking and comment. There are no answers to these in the course.

UNIT 1

Simple statements and commands

1.1	What does a computer do?	8
1.2	What is a computer?	8
1.3	What is BASIC?	10
1.4	A simple problem	10
1.5	Statement numbers	12
1.6	Execution and commands	14
1.7	Execution and data	16
1.8	INPUT, PRINT and LET	17
1.9	Store locations	19
1.10	Copying and overwriting	21
1.11	Arithmetic operators	23
1.12	Numerical constants	26
1.13	The remark statement: REM	27
1.14	More complicated arithmetic	28
1.15	Literal printing	28
	Assignment 1	30
	Objectives of Unit 1	31
	Answers to SAQ's and Exercises	31

1.1 What does a computer do?

In broad terms, a computer is a machine which helps us to solve certain kinds of problems. These usually involve symbols or characters which are familiar to us through every day use, e.g. letters of the alphabet (capital and lower case), numbers, punctuation marks and some special characters such as +, #, ★. The computer allows us to put in one set of symbols or characters and get out a different but related set. If this seems very vague and too general, let's consider some specific examples.

CHARACTERS IN	CHARACTERS OUT
Numbers representing the size of a window.	Cost of double glazing.
List of books borrowed from a library.	List of those books overdue.
A person's name.	The person's telephone number.
Standard notation for a move in a game of chess.	Picture of the chess board with the move accomplished.
Number representing height and acceleration.	Picture of lunar lander.
Pre-determined codes.	Musical sounds.

Figure 1 Some uses of a computer

This course is not about how the computer does these things but about how you can get it to do them by giving it the right instructions. We shan't therefore be going into any detail of the insides of a computer but you will find it helpful to know which are the major parts of a computer. This is quickly dealt with in the next section.

1.2 What is a computer?

A simple model of a computer is shown in Figure 2.

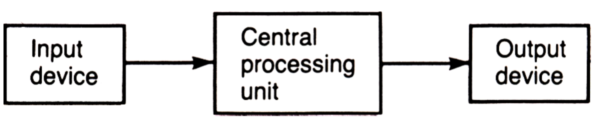


Figure 2 A simple model of a computer

You can see that there are three main parts to a computer:

1 The input device which allows you to enter either instructions or data (information) into the computer. On a microcomputer the input device is a keyboard which looks like a typewriter.

2 The central processing unit (CPU) which, amongst other things, carries out the instructions you have put in. This processing results in a modification of your data giving you the 'answer' or output that you require.
3 An output device which enables you to receive the result of the processing. The output device might be a television screen displaying the output or a printer which actually prints the output onto paper.

All this may sound very mundane. Indeed it would be were it not for the three key characteristics of a computer: (a) its capacity to store very large quantities of data which (b) it is able to process very rapidly and (c) its capacity to store a program which controls its own operation. This last characteristic is by far the most important one and is the one we are going to cover in this course.

Backing store

We shall just mention one other technical detail before looking at programming. If you are using this course you are likely to own or use a microcomputer with a small internal storage capacity. It uses that storage to keep its main running instructions plus the details of the problem it is currently solving. The latter details are erased when you switch the machine off so, if you want to keep your program or data, you have to keep them in a backing store: a separate storage system that you can link to the computer as needed. On small systems this will be an ordinary audio cassette tape and on larger systems, a magnetic disc store.

So, to summarise, the main elements of a computer system are illustrated in Figure 3.

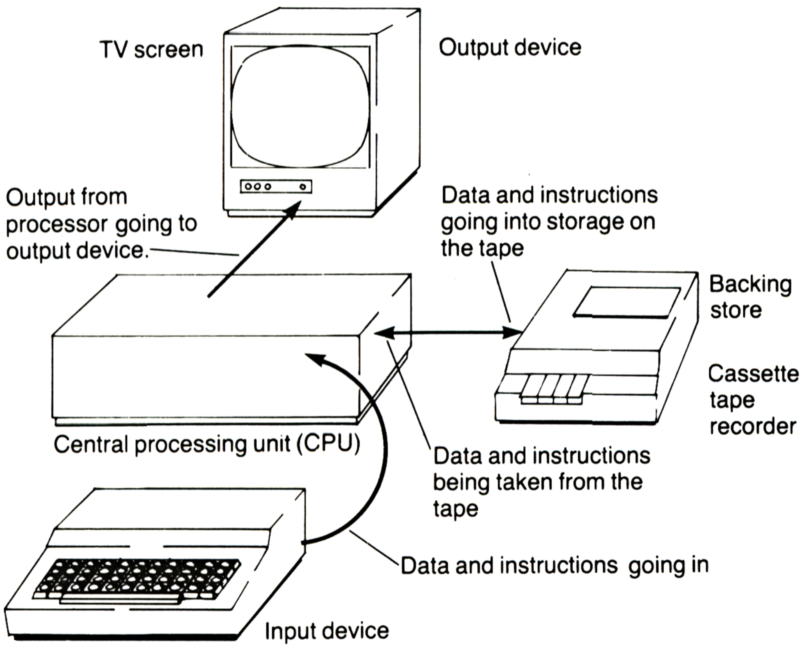


Figure 3 A typical computer system

1.3 What is BASIC?

A computer is an electronic device which processes patterns of electrical signals. If you had a problem, you wouldn't be able to feed it into the computer as electrical signals. Nor, if the computer was ready to give you an output, would you be able to understand it if it came as electrical signals. So the computer has a machine code inside it (put there by the manufacturer) to enable it to understand a programming code that you can understand. Machine codes are called low-level programming languages and correspond directly with the patterns of electrical signals in the computer. For obvious reasons, this program is called an interpreter. This course teaches you BASIC which is a high-level programming language You will then be able to use BASIC to program any computer that contains a BASIC interpreter. BASIC, by the way, stands for Beginners' All-purpose Symbolic Instruction Code.

You may find it useful to note the sequence of events that is taking place when you program a computer.

1 You have a problem.
2 You break down the problem into steps which can be put into BASIC.
3 You write the program in BASIC.
4 You sit at a keyboard and enter your program into the computer.
5 The computer interprets your BASIC instructions into its own code and processes them.
6 The computer prints out the results in the form you specified in the program.

And that is all you need to know about what a computer is. From now on we will assume that all you want to do is to give the computer problems and to get back results so now let's move on to a simple problem which we might want to give to a computer.

1.4 A simple problem

The main activity of programming is breaking down the solution to a problem into simple steps which can be represented by BASIC programming statements.

Imagine that you are playing the part of a computer with a young child. The child might give you two numbers and ask you to tell him their sum. After a short time the child will naturally try you out with large numbers which you cannot add in your head, so you will have to have a paper and pencil at hand. The following could be a typical dialogue.

CHILD: 'Start'
YOU: 'Give me the first number'
CHILD '12157'
(You write this number on a piece of paper)
YOU: 'Give me the second number'
CHILD: '7896'
(You write the second number on the piece of paper)
(You perform the addition sum)
YOU: '20053'

We could describe the computer's part in this process more formally in the following way:

1 Input the first number
2 Input the second number
3 Add the two numbers
4 Output the result

Figure 4 Computer processes in adding two numbers

By this simple analogy we have arrived at a strategy for solving this problem. Broadly speaking, phases 1 and 2 would be concerned with entering numbers into the computer, phase 3 would involve a process in the central processing unit, while phase 4 would involve the output device.

Now, although we have not taught you any BASIC programming yet, we are going to show you what the problem solving sequence would look like when written in BASIC.

Example 1
Write a BASIC program to enter two numbers into the computer and to output their sum.

Solution
We have already worked out an intuitive procedure to solve this problem in Figure 4. A program in BASIC would have the following form.

```
10   INPUT FIRST
20   INPUT SECOND
30   LET SUM = FIRST+SECOND
40   PRINT SUM
50   STOP
```

Program 1

We do not wish to concentrate on the details of the program at this stage, but hope that you can see, without stretching the imagination too far, how the strategy from Figure 4 has been changed into a program. A program then is a 'sequence of instructions composed for solving a given problem by computer'.

SAQ 1
We now come to the first point in the course at which we want you to check your progress through trying this Self-Assessment Question (SAQ). The SAQs are designed to help you find out whether or not you have understood the immediately preceding sections of the course. In each case, the answer to an SAQ appears at the end of the Unit in which the SAQ occurs. If you get all the answers right, just move onto the next section. If you get any wrong, check back to see where you have gone wrong.

Select those phrases from list B which complete correctly the phrases given in A.

A
1 The CPU . . .
2 The main characteristics of a computer system are . . .
3 A machine code is . . .

4 A machine code is an example of a . . . language.
5 BASIC is an example of a . . . language.
6 A BASIC interpreter . . .
7 A computer program is . . .

B
(a) low-level
(b) high-level
(c) . . . holds data and instructions, controls its own processing, and controls the operation of input and output devices.
(d) . . . a series of instructions or procedural steps for the solution of a specific problem.
(e) . . . that it is capable of storing large quantities of data, is able to process this data very rapidly and lastly that it is able to store a program which controls its own operation.
(f) . . . translates code written in BASIC into machine code.
(g) . . . a code which corresponds directly with the electrical patterns within a computer.

1.5 Statement numbers

Let's have a closer look at Program 1 again:

```
10  INPUT FIRST
20  INPUT SECOND
30  LET SUM = FIRST+SECOND
40  PRINT SUM
50  STOP
```

Program 1 (from p11)

We have said that a program is a sequence of instructions. In the program above each line is an instruction. Thus:

```
10  INPUT FIRST
```

is the first instruction of the program, and

```
50  STOP
```

is the last. Instructions in a programming language are sometimes called statements. We will use the two words synonymously.

Entering statements

When sitting at the keyboard of a micro, the actual process of entering a statement is completed only after the NEW LINE key has been pressed. So what happens is: You type 10 INPUT FIRST then press NEW LINE. Then you type 20 INPUT SECOND and press NEW LINE etc . . .

You see on the screen

```
10  INPUT FIRST
20  INPUT SECOND
```

You will have noticed that each line begins with a number. These must be whole numbers in the range 1–9999, and they determine the order in which the

instructions are processed (executed), i.e. they define the 'sequence' of the instructions. The execution of the instructions starts with the line of the lowest number, and continues in the sense of increasing numbers until instructed otherwise, or until the end of the program is reached. (More about 'until instructed otherwise' and 'ending', later.)

Why then, you may ask, was the program not written as follows?

```
1  INPUT FIRST
2  INPUT SECOND
3  LET SUM = FIRST+SECOND
4  PRINT SUM
5  STOP
```

Program 2

Why not indeed! The program would have done the job perfectly well! However, as you will soon find out when writing programs you need a certain amount of flexibility. In particular you need the opportunity to slip into the program a statement which you have overlooked, or one which will allow you to make an important modification. Numbering our lines 10, 20, 30 and 40 leaves 9 empty lines between statements which may be used to correct or modify the program. When running, the processing proceeds to the next highest line number of the program, so the gap of 9 unused line numbers does not slow down the program execution in any way.

SAQ 2

Look at the line numbers in the following programs and decide which programs would produce a correct sum of FIRST and SECOND.

(a)
```
11  INPUT FIRST
59  INPUT SECOND
93  LET SUM = FIRST+SECOND
401 PRINT SUM
500 STOP
```

(b)
```
23  INPUT FIRST
32  INPUT SECOND
49  LET SUM = FIRST+SECOND
40  PRINT SUM
50  STOP
```

(c)
```
10  INPUT FIRST
20  INPUT SECOND
15  LET SUM = FIRST+SECOND
40  PRINT SUM
50  STOP
```

(d)
```
100 INPUT FIRST
200 INPUT SECOND
110 LET SUM = FIRST+SECOND
190 PRINT SUM
220 STOP
```

(e)
```
100  INPUT FIRST
50   INPUT SECOND
407  LET SUM = FIRST+SECOND
902  PRINT SUM
1000  STOP
```

Programs 3–8

1.6 Executions and commands

The command RUN

Execution? No, it's not the end but the beginning! Let's get on and run our first program before we get tired of it!

```
10   INPUT FIRST
20   INPUT SECOND
30   LET SUM = FIRST+SECOND
40   PRINT SUM
50   STOP
```

Program 1 (from p11)

And what happens? Nothing. This is because the computer is waiting for us to give instructions to the program as a whole. If you want to execute this program, you must give it the command RUN. This you put on a new line as follows:

```
10   INPUT FIRST
20   INPUT SECOND
30   LET SUM = FIRST+SECOND
40   PRINT SUM
50   STOP
RUN
```

Program 1 (from p11)

(Don't worry about RUN not having a line number – we'll explain that shortly.)

Then press NEW LINE. You will then see L on the screen which is the computer's way of asking for data. Give it your first number; then press NEW LINE; another L appears because the computer needs your second number. Give it the second number and press NEW LINE. The answer should now appear. Here is our version of this run:

```
10   INPUT FIRST
20   INPUT SECOND
30   LET SUM = FIRST+SECOND
40   PRINT SUM
50   STOP
RUN
 12157
 7896
 20053
```

9/50 ———————————————————— means STOP at line 50

Figure 5   A complete run of a program

What we are doing therefore is to distinguish between the entry of a program and its execution. Let's go back to the dialogue between you and the child playing computers. A very explicit infant may have said 'I am going to give you two numbers, I want you to write them down, add them together, and then tell me their sum'. At this point you know exactly what to do, but you haven't yet done anything. You've got the instructions though, you've been programmed. The dialogue may proceed thus:

CHILD: 'Start'
YOU: 'Give me the first number'.
CHILD: '12157'
YOU: 'Give me the second number'.
CHILD: '7896'
YOU: '20053'

Now these instructions have been carried out (run). A program then is just a set of instructions for the computer. When the program is run or executed these instructions are carried out.

Other commands: LIST, SAVE, LOAD
RUN is not the only command which you can give to a program as a whole. You can also use LIST, SAVE and LOAD as follows.

LIST
For example, you may spend quite a lot of time typing a program into your machine and during that process make several corrections. You may then wish to see a fair copy of the program as a whole on the screen. If you type the word LIST, a complete copy of the program in line number order will appear on the screen. (With longer programs the report 5/0 appears when the screen is full. This means 'no more room on screen'. To see the next section of the program, key LIST followed by the last line number on the screen. Listing will then continue from this line.)

SAVE
Having developed a program to a satisfactory stage you may wish to take a copy of it on to tape or disc; the word SAVE (followed by the name you wish to give the program in quotes, e.g. SAVE "FRED") will do this for you. (Your computer will, of course, have to be connected to a back-up store such as a cassette tape recorder.)

LOAD
Later you may wish to use one of your stored programs; the command LOAD "FRED" will enter the program from your back-up store back into your computer.

Words like LIST, RUN, SAVE and LOAD which allow us to handle the program as a single entity are called commands and are provided by the BASIC interpreter. A command occupies a line on its own and generally does not have a line number, e.g. the word RUN after line 50 causes the program to start to execute, and is equivalent to the child's command 'start' in the dialogue above. Commands will be discussed in greater detail in a later unit, but the four that we have already looked at will allow us to get by for a start.

Computer response

K

After a command has been successfully carried out the interpreter informs the user of this fact by writing the message: K

(Not to be confused with our K in this course).

Keywords

These are the BASIC words which go in the program to specify what action is to occur. e.g. INPUT, PRINT, LET.

1.7 Execution and data

You will have realised that we have written a program which will add any two numbers in a quite general way, for it is immaterial to the program what numbers we enter when we receive prompts L on the screen. As the computer executes the program it must be able to request that we input actual numbers for its particular task, i.e. it must have the facility to demand specific data to do the job in hand. You need to be able to distinguish clearly between the program, as a set of more or less general instructions, and the data which are the actual numbers which must be input when the program is executing, in order to solve a particular problem. You can, of course, run your program repeatedly with many pairs of numbers, as you will see later.

Another way of looking at these instructions is to visualize the situation as an umpire gathers the runners at the start of a race in order to give them certain instructions: 'Go down the right-hand side of the field to the furthest corner, over the stile and turn left down the lane . . .' The umpire's instructions are analogous to a program. If the runners understand what he is saying then they know what to do; but they are still at the starting line. They haven't actually started. This is analogous to the program having been entered into the machine. Then the umpire says 'Go!' and the race starts. This is analogous to the computer starting to execute the program. Let's extend the analogy and consider the cross-country race as a novelty race. Imagine that the umpire did not give enough instructions for the runners to complete the course but he said something like 'When you get to the bottom of the lane you will find further instructions pinned to the oak tree . . .' These instructions should be sufficient to guide the runners over the next part of the course, i.e. on to the next clue, and so on until the end of the race. These clues are analogous to giving the program more data during the course of its execution. This analogy may help you to see the important distinction between entering a program into the machine, executing the program, and then inputting data during the course of its execution.

SAQ 3

Below is a print out from a computer. It contains keywords, commands, responses from the system and items of data. It also contains sections that are concerned with entry, execution and listing. Identify as many of these items as possible as follows:

Mark keywords with	K
Mark commands with	C
Mark system responses with	R
Mark data items with	D

Bracket lines concerned with entry
Bracket lines concerned with execution
Bracket lines concerned with list

```
10  INPUT FIRST
20  INPUT SECOND
30  LET SUM = FIRST+SECOND
40  PRINT SUM
50  STOP
RUN
[L] -37
[L] -46
-83

9/50
LIST

10  INPUT FIRST
20  INPUT SECOND
30  LET SUM = FIRST+SECOND
40  PRINT SUM
50  STOP

0/0
RUN
[L] 12.83
[L] 48.95
 61.78

9/50
RUN
[L] 17.0009
[L] -29.3629
-12.362

9/50
```

Program 1 (from p11)

1.8 INPUT, PRINT and LET

You have seen that a program is a sequence of statements, and we have given you an intuitive idea of how each statement works. You may also have noticed that each of the three types of statement used so far (INPUT, LET and PRINT) corresponds to one of the three main devices which comprise the computer system (input, central processor and output devices). We will now look at each statement in more detail.

INPUT

The word INPUT is a signal to the computer that during execution, an item of data must be entered at the input device. We saw this happen when we ran our first

program: after the $\boxed{L}$ we entered 12157, pressed NEW LINE to complete the input procedure and then found ourselves confronted by another $\boxed{L}$ requesting the input of the next number. What, then, happened to 12157, the first item of data? The answer is that it has been stored for later use in the program's execution in the storage location labelled FIRST. The word FIRST has two main functions in the program, (a) when written and later referred to by the programmer it reminds him that at this point in the program the first item of data should be input, and (b) when written in the statement 10 INPUT FIRST the word FIRST is the name or label of a location in the computer's memory. So 10 INPUT FIRST means enter a number at the input device and store it in the location labelled FIRST .

PRINT

The statement 40 PRINT SUM has almost the reverse effect to statements 10 and 20, in that it allows us to output information from the machine. It is a signal to the machine to take a copy of the contents of the store location labelled SUM and pass it to the output device which for most users of this course will be a television screen. Notice that LPRINT will literally result in a printed output if a printer is attached to your microcomputer but you still use the command PRINT when your microcomputer is attached to a television set for its output device.

LET

30 LET SUM = FIRST+SECOND is an example of an assignment statement. It is in this type of statement that the processing takes place. As you can see, it is a mixture of store names (SUM, FIRST, SECOND) and arithmetic operators (= and +). If you read it forwards, it says

> Let the store location SUM be made equal to the contents of the location FIRST added to the contents of the location named SECOND.

However, like many mathematical expressions, it is often clearer when read from right to left of the '=' as follows

> Add the contents of the location FIRST to the contents of the location SECOND and store the result in the location labelled SUM.

Generally the assignment statement has the form:
LET store location name = expression.

This means find the value of the expression of the right-hand side of the '=' and store this value in the store location named on the left-hand side of the '='.

The tricky point about LET . . . = . . . is that it is easily confused with . . . = . . . in mathematical equations. An example will demonstrate the difference. Suppose you have stored a number in location L and you want to make the number in that store 5 greater than it now is. You write:

LET L = L+5

Now obviously this doesn't mean:

L = L+5

since there is no value for L which could make this true. What it does mean is that the computer has added 5 to the number that is in store location L.

1.9 Store locations

As we have already said, one of the main characteristics of a computer system is its capacity for storing large quantities of data. We must now consider how the BASIC language allows us to allocate store location names. If you look at our first program and recall that a computer is capable of only doing one thing at a time, it is fairly obvious that when we reach line 20 and wish to input our second number, the number that we entered in line 10 must have been stored somewhere! In this case the first number was stored in the location labelled FIRST. We can think of the storage locations as being like a set of pigeon-holes where we distinguish clearly between the label or address or name on each pigeon-hole and the contents of the hole.

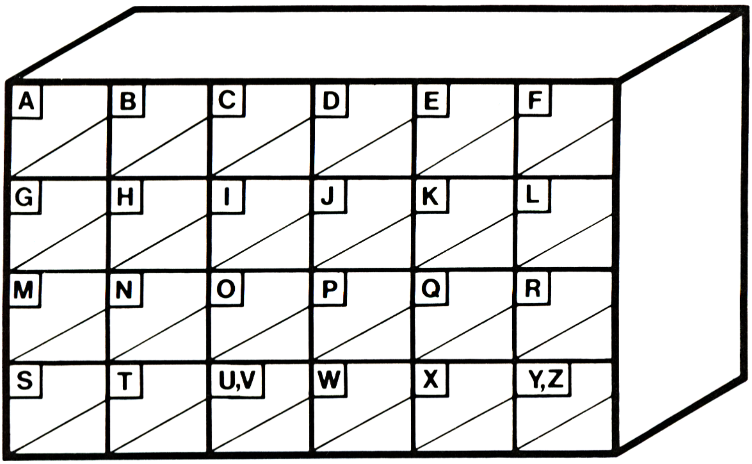


Figure 6 A model of the store locations in a very small computer

You will see that in our model of possible store locations we have used the labels A, B, C . . . On the other hand, in the program we have been studying we have used words of up to six characters to label our stores, e.g. FIRST, SECOND. This brings us to our first major point of difference between the interpreters for the various micro-computers now available. Some BASIC interpreters allow a much wider range of storage names than others. Some machines limit your variable name to a few characters. Others allow longer variable names or even names of unrestricted length. The ZX81 allows variable names of unlimited length.

Choosing store location names

Clearly it makes life a lot easier for the programmer if he chooses store location names which remind him of what he is storing. That is why we chose FIRST, SECOND and SUM. We could have used A, B and S so that our program would have been:

```
10  INPUT A
20  INPUT B
30  LET S = A+B
```

```
40  PRINT S
50  STOP
```

Program 9

🄺 When you see this symbol it means we suggest you try this program on your own microcomputer if you have one. To do this, key in the lines, press NEW LINE and then press RUN. Your computer will ask you for a number. Give it one and press NEW LINE. It then asks you for the second number. Give it the second number, press NEW LINE and the sum will appear on your screen.
If we had done this, then after inputting 12157 and 7896, the store locations would be:

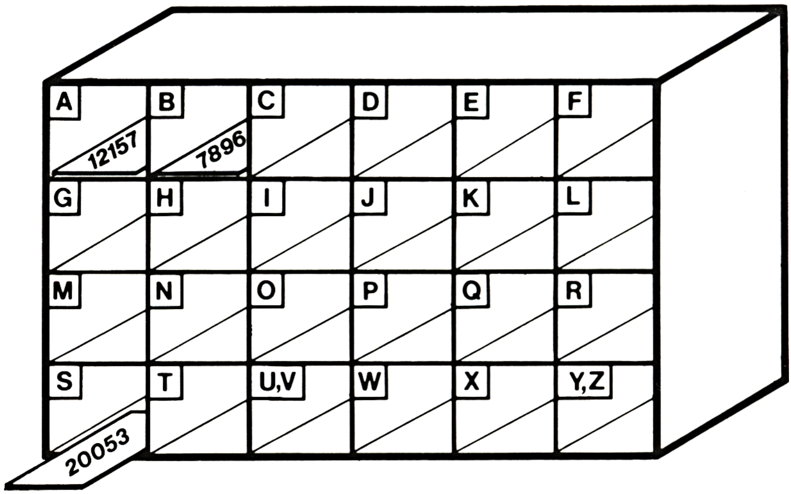


Figure 7 State of the store locations after Program 9.

The common system of location names

Since long store location names are not available on many microcomputers, we will, from now on, use the system of location names that works on practically every microcomputer until we come to lists in a later unit. The system we shall use labels a location by a capital letter followed by a digit. This gives us 286 possible locations as shown in Figure 8.

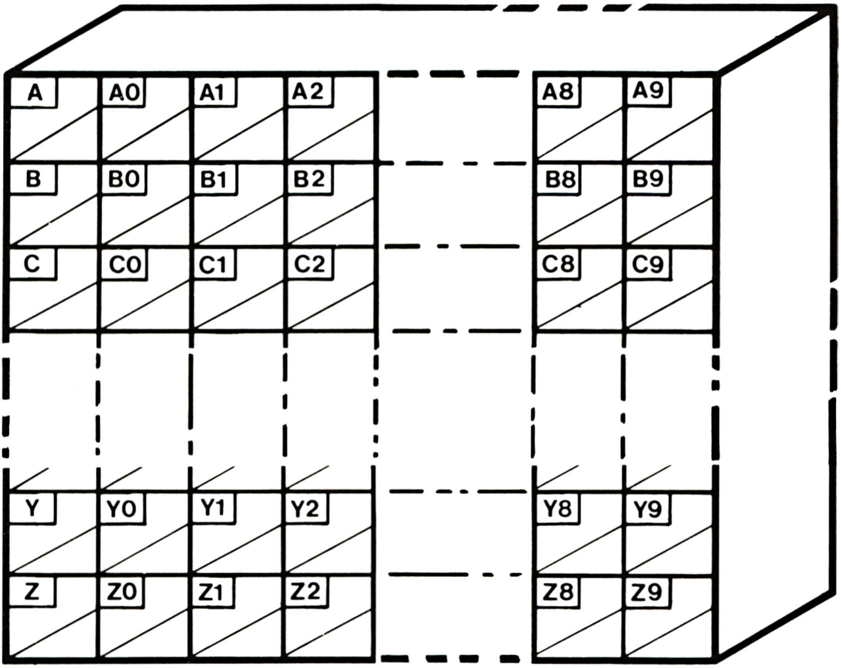


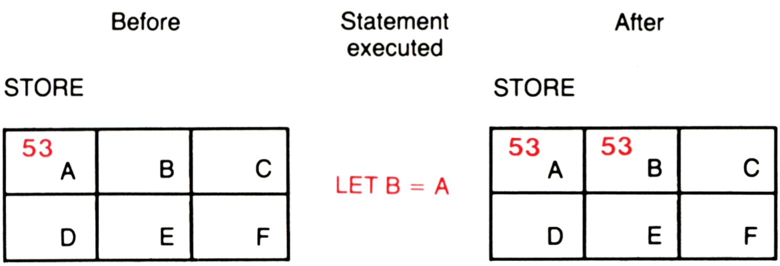
Figure 8 286 possible store locations

1.10 Copying and overwriting

BASIC statements can have two different effects on the contents of a store location. Or rather a statement can have no effect on the contents of the location or it can change the contents. This is illustrated below.

Effect of copying

Suppose we have the number 53 stored in location A. What happens after LET B= A and after PRINT A? In each case A still stores the number 53 after the statement has been executed:



Before	Statement executed	After	Monitor
STORE		STORE	
53 A · B · C / D · E · F	PRINT A	53 A · B · C / D · E · F	53

STORE (Before)

53 A	B	C
D	E	F

Statement executed: PRINT A

STORE (After)

53 A	B	C
D	E	F

Monitor: 53

In each case the copying statements leave the original store location unchanged. It's just like getting a statement of your bank account: the piece of paper copies your account but your account still has your money in it!

Effect of overwriting

Suppose now that we still have the number 53 stored in location A but this time execute the statement LET A = A+7. The result is:

Before	Statement executed	After
STORE		STORE

STORE (Before)

53 A	B	C
D	E	F

Statement executed: LET A = A+7

STORE (After)

60 A	B	C
D	E	F

The statement LET A = A+7 overwrites the contents of A. That is, the original contents disappears and is replaced by the new contents, which is in this case, 60. Of course we could have made the new contents of A to be 60 in many ways, e.g. by, say: LET A = 60.

SAQ 4

Which of the following are valid store location names for numbers, according to the rules given on page 20.

(a) N3
(b) 3N
(c) W10
(d) B#
(e) QJ
(f) M
(g) M5
(h) M-5
(i) M+5
(j) U0

Give reasons for discarding those names which you reject.

1.11 Arithmetic operators

When you do arithmetic you use four main operators: +, −, × and ÷. BASIC has the same operators, although two are printed differently:

Every day symbol	Meaning	BASIC symbol
+	add	+
−	subtract	−
×	multiply	★
÷	divide	/

SAQ 5
Write the following expressions using BASIC symbols for the arithmetic operators. (Where the expressions use brackets, leave the brackets in your answers.)

(a) 3+7
(b) 3×7
(c) 8÷4
(d) 5×(2+8)
(e) 30÷(3+2)
(f) 24−(4×3)
(g) 5×6×7
(h) 81−(27×2)

SAQ 6
If A has the value of 2, B has the value of 5 and C has the value of 10, calculate the values of the following:

(a) A+B+C
(b) A★B
(c) A★B★C
(d) C/A
(e) C/(B−A)
(f) A★A
(g) (B★C)/(B−A)
(h) (C−B)★(C+B)

The arithmetic is actually done (executed) in BASIC through assignment statements (LET statements) which tell the computer's arithmetic unit (part of the central processor) what to do. We can illustrate this with the following computer model.

Effect of LET A = B−C

STORE AT START

A	15 B	10 C
D	E	F

LET A = B−C

Remember to read this from the right to the left of the '=' sign. It says take the number in location B, subtract from it the number in location C and put the result in location A. So the result is:

Result of LET A = B−C

Notice that the contents of B and C are unchanged.

STORE AT FINISH

5 A	15 B	10 C
D	E	F

Effect of LET A = B★C

STORE AT START

A	15 B	10 C
D	E	F

STORE AT FINISH

150 A	15 B	10 C
D	E	F

SAQ 7
Fill in the values in the store locations A, B and C after each line has been executed in these programs.

1. Program — Store location values

Program	A	B	C
10 LET A = 12			
20 LET B = 5			
30 LET C = A★(A+B)			
40 LET A = A+10			

2. Program — Store location values

Program	A	B	C
10 LET A = 20			
20 LET B = A★3			
30 LET C = A/4			
40 LET A = B+C			

What you have just done isn't (we hope) difficult and it doesn't get any more difficult when we move on to more complicated store location names. We have to make the names more complicated because A, B, C . . . only gives us 26 stores and, as we said on page 21, we are going to use location names A, A0, A1, etc. So,

LET P4 = Q1★R1

is no different from LET A = B★C. P4, Q1 and R1 are simply store location names. P4 is one name, just as XYZ 823A is one car number.

We are now ready to use the arithmetic capacity of a computer.

Example 2
Write a BASIC program to enter two numbers into the computer and then output their sum, difference, product and quotient.

Solution
This may look complicated but we've really solved this already. We had a program (*Program 9*) to output the sum of two numbers, so to output their difference, product and quotient, we only need to change the arithmetic operator in line 30 which reads

```
30  LET SUM = FIRST+SECOND
```

First, however, we will rewrite the program using the shorter store location names:

Original version	**New version**
10 INPUT FIRST	10 INPUT N1
20 INPUT SECOND	20 INPUT N2
30 LET SUM = FIRST+SECOND	30 LET S = N1+N2
40 PRINT SUM	40 PRINT S
50 STOP	50 STOP

Program 10

Now what we need is three extra versions of the new version, each with a different line 30:

10 INPUT N1	″ ″	″ ″	″ ″
20 INPUT N2	″ ″	″ ″	″ ″
30 LET S = N1+N2	″ D = N1−N2	″ P = N1★N2	″ Q = N1/N2
40 PRINT S	″ D	″ P	″ Q

(Using D for the location for difference, P for the location for product and Q for the location for quotient.)

Do we need to write four programs? Fortunately no, because when we copy the numbers from locations N1 and N2, we don't destroy these contents so we can use them four times over in one program:

```
10  INPUT N1
20  INPUT N2
30  LET S = N1+N2        original program for sum
40  PRINT S
50  LET D = N1−N2        extra lines for difference
60  PRINT D
70  LET P = N1★N2        extra lines for product
80  PRINT P
90  LET Q = N1/N2        extra lines for quotient
100 PRINT Q
110 STOP
```

Program 11   Sum, difference, product and quotient of two numbers

[K] Key Program 11 into your microcomputer. Then key RUN and input two numbers. A typical print out should look like:

```
RUN
[L] 57.82
[L] 19.11
 76.93
38.71
 1104.94
 3.02564
```

1.12 Numerical constants

Earlier in this unit we saw that our first BASIC program was capable of manipulating whole, decimal and negative numbers. At this stage we won't go into detail on how numbers are represented in BASIC, but just show you that we can use numbers directly in assignment statements.

The statement LET P = 427★R means create the number 427, multiply it by the number found in store location R and then store the result in location P. (Don't be put off by thinking that computers only handle binary numbers. The computer's interpreter enables us to input ordinary decimal numbers.)

Similarly, the statement LET Y4 = 3.142+Z8 creates the number 3.142 and adds it to the contents of location Z8, and then stores the sum in location Y4.

And the statement LET A = −48.93/B creates the number −48.93 and divides it by the number found in location B and then stores the result of this calculation in location A.

Exercise preamble
Progress in metrication has been slow and life still abounds with irritating little conversions which occasionally tease us, e.g. pounds weight to kilogrammes, yards to metres, pints to litres, a knitting pattern with balls of wool in ounces which must be bought in grammes, etc. . . . If we go on holiday we mentally convert kilometres into miles, and pounds sterling into other currency. Most of us still think of body and weather temperatures in terms of degrees Fahrenheit rather than Centigrade or Celsius. We can imagine the home microcomputer of the future having a general conversion program in it which will do all these diverse conversions for us. The next two exercises are on writing programs to do conversions. You only need the ideas introduced in the earlier programs in this Unit.

Exercise 1
Write programs in BASIC to carry out each of the following conversions:

(a) Input a number representing a length in inches, and output this length in centimetres, given that one inch is equivalent to 2.54 centimetres.

(b) Input a number representing a weight in ounces, and output that weight in grammes, given that one ounce is equivalent to 28.375 grammes.

(Answers to Exercises appear at the end of each unit with the SAQ answers.)

Exercise 2
Any conversion involves 'conversion factor × number to be converted' so it is possible to write a general conversion program where you input two numbers each time you use it: the conversion factor and the number to be converted. Write a general conversion program which will do this.

1.13 The remark statement: REM

The statement REM is the remark statement. It allows us to give a title to a program or make some other meaningful remark about the program. For example, within the body of a program it helps us identify what the program or section of the program does. The REM statement is not executed by the computer and is there purely for the benefit of either the programmer or user, i.e. when the computer sees REM at the beginning of a line it ignores everything on that line. The next program is concerned with calculating percentages and so as a title to the program our first statement will be 10 REM ★★PERCENTAGE CALCULATION★★. (The ★★ have no function other than to emphasise the title.)

Example 3
Write a BASIC program to input two numbers and output the second as a percentage of the first.

Reminder Percentage = (second÷first)×100.

Solution

```
10  REM★★PERCENTAGE CALCULATION★★  (program title using
                                    REM statement)
20  INPUT F
30  INPUT S
40  LET P = (S/F)★100
50  PRINT P
60  STOP
```

Program 12 Percentage calculation

Typical runs

```
RUN
[L] 57
[L] 74
 129.825

9/60
RUN
[L] 74
[L] 57
 77.027

9/60
```

[K] Program 12.

1.14 More complicated arithmetic

We have reached the stage when we can use the computer like a simple four-function calculating machine, but we will soon wish to do slightly more complicated arithmetic. Generally BASIC allows us to set out equations in a familiar way. We can use brackets, i.e. ( ) to group together certain values, and when BASIC evaluates an expression it deals with the values *inside* the brackets first. Next come values involving multiplication or division and finally, addition and subtraction.

This order of preference for performing arithmetic operations is discussed more fully in a later unit, but you will soon see that this order just formalises the way we naturally go about arithmetic calculations.

Let's show you what we mean.

Example 4

Write the following expressions in BASIC:

1. ab+c   2. a(b+c)   3. $\frac{a}{b+c}$

Solutions

1. A★B+C

   The order of precedence rules tell us that A★B will be evaluated first and the C added. If you were worried about this you could write (A★B)+C but the brackets aren't essential here.

2. A★(B+C)

   Notice that, just as a bracket is needed in a(b+c) so it is needed in A★(B+C).

3. A/(B+C)

Now try some for yourself.

SAQ 8

Write the following as BASIC expressions:

1. abc
2. $\frac{ab}{c}$
3. $\frac{a+b}{c}$

Exercise 3

Now that you have written the expressions in SAQ 8 as BASIC expressions, write a program that will allow you to input three numbers (A, B and C) and print out the values of the expressions in SAQ 8.

1.15 Literal printing

You have seen already that we can print out the values from store locations. You will find as the course progresses that the PRINT function is very versatile. One use of this statement is to print messages on the monitor screen which will be helpful to the user when the program is running. These messages are usually referred to as prompts. We have seen already that when an input statement is

encountered during the execution of a program, a $\boxed{L}$ appears on the screen to remind us that an input is required. In even slightly complicated programs, a series of $\boxed{L}$ s on the screen is confusing to the user since he may not know which input value the $\boxed{L}$ is prompting. Prompts generated by PRINT statements are very useful in these circumstances.

It is very easy to get a computer to print a reminder or message on the screen. All you need is a line such as:

```
20   PRINT "MESSAGE"
```

This simply prints

```
MESSAGE
```

on your screen.

In other words, whatever appears between quotes thus " " after the word PRINT will be printed out exactly as it stands. Notice that, as in the case of the REM statement, the computer doesn't execute the words in the quotes. Thus

```
20   PRINT "A+B"
```

results in

```
A+B
```

on your screen and the computer does not add the value in location A to the value in location B.

The following example demonstrates the use of PRINT " " to remind the programmer and user of what the program is doing.

Example 5
Write a BASIC program to convert a temperature value given in degrees C into degrees F.
Remember $°F = \frac{9}{5} \times °C + 32$

Solution

```
10   REM★★CENTIGRADE TO FAHRENHEIT★★
20   PRINT"ENTER NEXT TEMP IN DEGREES C"
30   INPUT C
35   PRINT C
40   LET F = (9/5)★C+32
50   PRINT"THIS TEMP IN DEGREES F IS"
60   PRINT F
70   STOP
```

print message precedes input statement so that the message is printed before $\boxed{L}$ appears

Program 13 Temperature conversion

Typical run

```
RUN
ENTER NEXT TEMP IN DEGREES C
 16
THIS TEMP IN DEGREES F IS
 60.8

9/70
```

$\boxed{K}$ Program 13.

Assignment 1

NEC students: your solution to this assignment should be sent to your NEC tutor for marking. If you own your own microcomputer, write your own assignment in the BASIC for that computer and tell your tutor what make of microcomputer it is. FlexiStudy students: complete the assignments according to the instructions given to you by your FlexiStudy centre.

Remember to make good use of remark and literal print statements when writing your programs.

1. If you deposit £D in an account paying P% rate of interest for one year, then the yield at the end of the first year is given by the equation

$$Y = D \times \frac{P}{100}$$

(a) Write a BASIC program to input values for D and P and to output the yield, Y.
(b) If the original deposit together with the accrued interest is left in the account for a further year at the same rate of interest then the compound interest after the second year will be given by the equation

$$C = (D+Y) \times \frac{P}{100}$$

Extend your program for (a) to calculate and output this compound interest.

2. Consider the problem of estimating the cost of installing replacement aluminium double glazed windows. The windows comprise 3 parts:

(a) a hardwood surround, (b) aluminium frame and (c) glass. If the height of the window is H metres and the width is W metres, then the total lengths of both hardwood and aluminium required are given approximately by the expression (2H+2W) metres; and the area of glass required by the expression (H×W) square metres.

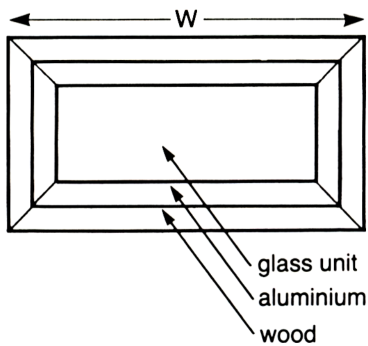


Figure 9

Write three separate BASIC programs which, on being given values for height and width in metres, will output the cost of
(a) the hardwood surround if the wood costs £3 per metre;
(b) the aluminium surround if the aluminium costs £4 per metre;
(c) the glass unit if the glass unit costs £40 per square metre.

Now link these three into one program to estimate and output the total cost of installation, if the labour cost is £50 per window.

Objectives of Unit 1

Now that you have completed this Unit, check that you are able to:

Write simple programs using:

Line numbers	☐
INPUT	☐
LET	☐
PRINT	☐
Store locations identified by a single letter or a letter followed by a single digit	☐
Copying from one location to another	☐
Overwriting	☐
+, −, ★, /	☐
( )	☐
Numerical constants	☐
REM	☐
PRINT " "	☐

Know when to use:

NEW LINE	☐
RUN	☐

Know how to respond to:

number/number	☐
$\boxed{L}$ $\boxed{K}$	☐

Answers to SAQ's and Exercises

SAQ 1

A	B
1	(c)
2	(e)
3	(g)
4	(a)
5	(b)
6	(f)
7	(d)

SAQ 2

(a) and (e) would run as Program 1.
(b) is asked to print SUM before SUM has been calculated.
(c) and (d) are asked to calculate SUM before SECOND has been inputted.

SAQ 3

```
                 (K)
       10  INPUT FIRST
       20  INPUT SECOND                  ]
       30  LET SUM = FIRST+SECOND        ]  entry
       40  PRINT SUM                     ]
K ——   50  STOP
C ——   RUN
R ——   [L]-37 ——————————————— D
```

```
R ——— [L] -46 ——————————————————— D  ]
      -83 ——————————————————————— R  ]  execution

R ——— 9/50
C ——— LIST            (K)

      10  INPUT FIRST                   ]
      20  INPUT SECOND                  ]  listing
      30  LET SUM = FIRST+SECOND        ]
      40  PRINT SUM                     ]
K ——— 50  STOP

R ——— 9/50
C ——— RUN
R ——— [L] 12.83 ————————————————— D  ]
R ——— [L] 48.95 ————————————————— D  ]  execution
       61.78 ———————————————————— R  ]

R ——— 9/50
C ——— RUN
R ——— [L] 17.0009 ——————————————— D  ]
R ——— [L] -29.2629 —————————————— D  ]  execution
      -12.362 ——————————————————— R  ]

R ——— 9/50
```

(Have you noticed that this program has coped with negative and decimal fractional numbers?)

SAQ 4

(a) OK
(b) No, begins with a digit instead of a letter.
(c) Not allowed on all systems. (10 is two digits, not 1 as in, say, W8.)
(d) No, # is not an acceptable symbol in a variable name.
(e) No, uses two letters. (This will, of course, work on some machines.)
(f) OK
(g) OK
(h) No, '-' is not an acceptable symbol.
(i) No, '+' is not an acceptable symbol.
(j) OK

SAQ 5

(a) 3+7 (b) 3★7 (c) 8/4 (d) 5★(2+8) (e) 30/(3+2)
(f) 24−(4★3) (g) 5★6★7 (h) 81−(27★2)

SAQ 6

(a) 17 (b) 10 (c) 100 (d) 5 (e) 10/3 or 3⅓ or 3.33 . . .
(f) 4 (g) 50/3 or 16⅔ or 16.66 . . . (h) 75

SAQ 7

1.

12		
12	5	

2.

20		
20	60	

$\boxed{12}$ $\boxed{5}$ $\boxed{204}$ $\boxed{20}$ $\boxed{60}$ $\boxed{5}$

$\boxed{22}$ $\boxed{5}$ $\boxed{204}$ $\boxed{65}$ $\boxed{60}$ $\boxed{5}$

Exercise 1

(a) Program 14

```
10  INPUT L1
20  LET L2 = 2.54★L1
30  PRINT L2
40  STOP
```

Typical runs

```
RUN
[L] 12        first use
 30.48

9/40
RUN
[L] 36        second use
 91.44

9/40
```

$\boxed{K}$ Program 14.

(b) Program 15

```
10  INPUT W1
20  LET W2 = W1★28.375
30  PRINT W2
40  STOP
```

Typical runs

```
RUN
[L] 10        first use
 283.75

9/40
RUN
[L] 50        second use
 1418.75

9/40
RUN
[L] 16        third use
 454

9/40
```

$\boxed{K}$ Program 15

Exercise 2

Program 16

```
10  INPUT V
20  INPUT F
30  LET N = F★V
40  PRINT N
50  STOP
```

Typical runs

```
RUN
[L]16         use for ounces
[L] 28.375    to grammes
 454

9/50
RUN
[L] 36        use for inches
[L] 2.54      to cms.
91.44

9/50
```

$\boxed{K}$ Program 16

SAQ 8

1. A★B★C
2. A★B/C[(A★B)/C is also correct]
3. (A+B)/C

Exercise 3

Program 17

```
10  INPUT A
20  INPUT B
30  INPUT C
40  LET R = A★B★C        calculates first expression and prints it out
50  PRINT R
60  LET R = (A★B)/C      calculates second expression and prints it out
70  PRINT R
80  LET R = (A+B)/C      calculates third expression and prints it out
90  PRINT R
100 STOP
```

Notice that we can use R as the location for all three answers because we copy (print out) each answer in turn before we overwrite with the next answer.

Typical runs

```
RUN                      RUN
-19375.1                  2197
-6.3586957                13
-0.25362319               2

9/100                    9/100
```

[K] Program 17

UNIT 2

Making decisions

2.1	Introduction	36
2.2	PRINT . . . ,	36
2.3	Repetitions and GOTO	38
2.4	Programming style	40
2.5	IF . . . THEN GOTO . . .	40
2.6	Inequalities	42
2.7	Flowcharts	45
2.8	Counting	48
2.9	Comparisons	53
	Assignment 2	54
	Objectives of Unit 2	54
	Answers to SAQ's and Exercises	55

2.1 Introduction

The programs which we considered in Unit 1 were quite straightforward. They started their processing at the statement with the lowest number and continued in line number order until execution finished at the line with the highest line number. One thing that computers are very good at is lots of repetitive calculations; another is their ability to make decisions. Both of these features involve changing the sequence in which a program is executed. This Unit will introduce you to some of the statements which enable you to write programs of this type. But first we are going to introduce you to a new type of PRINT statement.

2.2 PRINT . . . ,

In Unit 1 we wrote a program (Example 3) to output one number as a percentage of another. On the screen, the calculation and result appeared in the format:

RUN
$\boxed{\text{L}}$ 57
$\boxed{\text{L}}$ 74
129.825

Obviously it would be better if the answer included the word Percentage so that it was clearer what was happening. This can easily be done by changing line 50 from 50 PRINT P to

```
50  PRINT "PERCENTAGE", P
```

The effect of this is:

Version of line 50	**Result on screen**
50 PRINT P	129.825
50 PRINT "PERCENTAGE", P	PERCENTAGE 129.825

The statement PRINT “PERCENTAGE”, P has four items in it.

PRINT | “PERCENTAGE” | , | P

P — the variable whose value will be printed

, — comma tells the computer to print P at the 16th character position across the page

“PERCENTAGE” — “ ” identifies the word(s) to appear on the screen

PRINT — instructions to print

We can use PRINT . . ., to improve the percentage program from Unit 1. At the same time we can improve the appearance of the program by making use of the literal print statement PRINT “ ” which we introduced in section 1.1.5.

Original program

```
10  REM★★PERCENTAGE CALCULATION★★.
20  INPUT F
21  PRINT F
30  INPUT S
31  PRINT S
40  LET P = (S/F)★100
50  PRINT P
60  STOP
```

New program

```
10  REM★★PERCENTAGE★★
20  PRINT "INPUT THE FIRST NUMBER"
30  INPUT F
31  PRINT F
40  PRINT "INPUT THE SECOND NUMBER"
50  INPUT S
51  PRINT S
60  LET P = (S/F)★100
70  PRINT "PERCENTAGE",P
80  STOP
```

Program 1 Improved percentage program

Original program: typical run

```
RUN
[L]80
[L]37
  46.25
```

New program: typical run

```
RUN
INPUT THE FIRST NUMBER ———— effect of line 20
[L] 80
INPUT THE SECOND NUMBER ——— effect of line 40
[L] 37
PERCENTAGE        46.25 ———— effect of line 70
                  └————————— 15th position
```

Notice how the use of literal print at lines 20 and 40, together with PRINT " ", and the printing of inputted data makes the program much more friendly and understandable when in use.

[K] Program 1.

Print zones
There are two print zones on a ZX81 screen. Zone 1 is on the left of the screen. Zone 2 is given by the use of the comma in PRINT. Thus

PRINT "ZONE 1", "ZONE 2"

results in

ZONE 1   ZONE 2

and

PRINT "PERCENTAGE", P gave PERCENTAGE   46.25

whereas

PRINT "PERCENTAGE";P would give PERCENTAGE46.25

SAQ 1
What would appear on the screen as a result of these print lines?
(Assume A=48, B=8, C=6 in these questions.)

(a) PRINT "AREA";A
(b) PRINT "LENGTH";B, "WIDTH";C
(c) PRINT "LENGTH", "WIDTH"
PRINT B, C

Write PRINT lines in BASIC which would print on the screen the following words in the zones shown:

	Zone 1	Zone 2
(d)	LENGTH	8
(e)	LENGTH	8
	WIDTH	6
	AREA	48
(f)	LENGTH	WIDTH

2.3 Repetitions and GOTO

Suppose now that you wanted to use Program 1 to calculate all the percentages for a test taken by a whole class of pupils. You would have to use the program over and over again, starting at RUN each time, e.g.:

```
RUN
INPUT THE FIRST NUMBER
  80
INPUT THE SECOND NUMBER
  42
PERCENTAGE          52.5

9/60
RUN
INPUT THE FIRST NUMBER
  80
INPUT THE SECOND NUMBER
  19
PERCENTAGE          23.75

9/60
```

Figure 1 Repeated use of percentage program

It would be much easier if, after the computer has calculated the first percentage, it went back to the beginning of its calculation and asked us for the next mark. This we can make it do using the statement

GOTO line number

which redirects the program to whichever line number we insert. Here is the percentage program re-written in this way:

```
10  REM★MARKS INTO PERCENTAGES★
20  PRINT "INPUT THE TOTAL POSSIBLE MARKS"
30  INPUT T
31  PRINT T
40  PRINT "INPUT THE NEXT MARK"
50  INPUT M
51  PRINT M
60  LET P=(M/T)★100
70  PRINT"PERCENTAGE",P
80  GOTO 40
90  STOP
```

Program 2 Percentage program for repeated use

Notice the new lines 20 and 30 which ensure that we only have to enter the maximum mark on the test once. The calculation is carried out in lines 50 to 70 and, in reading line 80, the program returns to line 40 to ask us for another mark. A typical run is:

```
RUN
INPUT THE TOTAL POSSIBLE MARKS
  80
INPUT THE NEXT MARK
  42
PERCENTAGE       52.5
INPUT THE NEXT MARK
  67
PERCENTAGE       83.75
INPUT THE NEXT MARK
  19
PERCENTAGE       23.75
  55
PERCENTAGE       68.75
INPUT THE NEXT MARK
```

Remember, if the screen fills, then press CONT to carry on with the program.

The GOTO statement interrupts the program's normal execution in line number order. As soon as the program reads GOTO, it unconditionally transfers control to the line number in the statement. It is sometimes called an 'unconditional jump'.
[K] Program 2. (Press STOP, then NEWLINE when you are fed up with it!)

SAQ 2
The following program squares numbers (i.e. multiplies a number by itself). Add a GOTO line to allow you to use the program over and over again to square successive numbers.

```
10  INPUT N
20  LET S = N★N
30  PRINT S
40  STOP
```

Program 3 Calculating squares

[K] key and run lines 10 to 40.
Add your extra line and run your new program.

2.4 Programming style

The use of the GOTO statement helped in some ways. However, it still left some loose ends, such as the [L] at the end of the run. On reaching line 80 in Program 2 control is always returned to line 40 which then generates the demand for a further input; hence the [L]. The program then is locked in a perpetual loop from which it cannot escape. The only way to stop this program is to break out of it which is accomplished by pressing STOP

To have the execution of the program left as it were in mid-air is obviously bad style, but we will sort this out in a little while. More importantly, we ought to warn you of the dangers of using GOTO. As we have said, this statement allows you to jump or branch to virtually any position in the program, which may at this stage seem to be a useful facility. But because GOTO allows us to jump rather at random to any point in the program, it is often used in such a way that the logical structure of the solution is broken up by 'jumps of convenience' to other parts of the program rather than by following the logical structure of the analysis of the problem. We will, therefore, use the GOTO statement sparingly throughout this course. We shall only use it when we think that the clarity of the program will be marred if it is not used. We hope that you will also try and follow our example and use the GOTO statement as little as possible. Though we will avoid its indiscriminate use, you will find it more widely used in some text-books and computer magazines.

2.5 IF . . . THEN GOTO . . .

The problem that we have just left is how to signal to the computer that we have reached the end of the list of marks. When doing a manual calculation, we can see that we have reached the end, or, if not, we would have carried out some sort of counting procedure. In a little while we will see how the computer may be used to count for us, but first we will introduce a means of signalling that the end of the list has been reached.

One method is to end the list of numbers that you put in with a special number that will 'stick out like a sore thumb', eg −9999. We would hardly expect a pupil to have

obtained −9999 marks in any test! This value is called a dummy or terminating value, or sometimes a rogue value. We want the program to run as normal when 'proper' marks are inputted but to stop when the mark −9999 is inputted. In other words, we want to be able to write a program with the following logical structure:

1. Start.
2. Input the total marks.
3. Input the next mark.
4. If this value is equal to −9999 then go to line 8 otherwise carry on to line 5.
5. Calculate the percentage.
6. Output the percentage.
7. Go to line 3.
8. Stop.

Figure 2 Stopping the percentage calculation

Fortunately there is a BASIC statement which will carry out the decision in line 4. It is:

IF . . . . . . THEN GOTO line number

condition to be satisfied for program to jump to given line number

So all we need to do is to translate statement 4 in Figure 2 as

```
40 IF M = -9999 THEN GOTO 80
```

and insert it into Program 4. This statement means: if the value found in M is equal to −9999, then go to line 80, otherwise continue executing the next statement after 40. The statement in Figure 2 'otherwise carry on to line 5' is not translated into BASIC but is implied: either jump out of sequence or carry on in sequence. We can do this very conveniently by using some new line numbers between the ones we have been using. (You will remember that we deliberately spread out statements 10, 20, 30 . . . so as to leave room to put extra statements in later.) This gives us:

```
10 REM★★PERCENTAGES★★
20 PRINT"INPUT THE TOTAL MARKS"
25 INPUT T
26 PRINT T
30 PRINT "INPUT THE NEXT MARK"
35 INPUT M
36 PRINT M
40 IF M = -9999 THEN GOTO 80
50 LET P = (M/T)★100
60 PRINT"PERCENTAGE",P
70 GOTO 30
80 STOP
```

Program 4 Percentage calculation with a terminating value

You use Program 4 in exactly the same way as Program 2 until you have put in the last mark to be converted to a percentage. Then you enter the mark −9999 and the program ends.

Here is a typical run.

```
RUN
INPUT THE TOTAL MARKS
   80
INPUT THE NEXT MARK
   43
PERCENTAGE          53.75
INPUT THE NEXT MARK
   29
PERCENTAGE          36.25
INPUT THE NEXT MARK
   62
PERCENTAGE          77.5
INPUT THE NEXT MARK
   -9999 ———————————————— terminating mark

9/80
```

[K] Program 4 and use it to convert some marks of your own. Terminate your run with −9999.

SAQ 3
You ended SAQ 2 with the program:

```
10  INPUT N
20  LET S = N★N
30  PRINT S
35  GOTO 10
40  STOP
```

but like the percentage program, this never stops. Modify the program to include a dummy value for terminating the program.

2.6 Inequalities

It was helpful to be able to use the expression M = −9999 in the IF . . . THEN GOTO . . . statement to determine whether or not the branch should occur. The statement means if M = −9999 is true then branch, otherwise carry on. The = states a relationship between M and −9999.

BASIC allows expressions to include relationships:

Relationship	**Example**	**Meaning**
>	A>B	the value in store location A is greater than the value in B
<	X<Y	the value in store location X is less than the value in Y

True or false?

Consider the expression A<B. If A = 2 and B = 5 then A<B is true, because 2 is less than 5. Consider the same expression with values A = 2 and B = 1. Now A<B is false, because 2 is not less than I. Similarly if A = 2 and B = 2 then A<B is false, for 2 is not less than 2. In writing programs we often find it useful to be able to know whether a statement involving = or < or > will be true or false. This is called the logical state of the assertion, e.g.

Assertion	**Logical state**
3>2	True
7<7	False

You will probably find this easy enough for positive whole numbers but may be less sure of what happens in other cases. If in doubt, remember the number line:

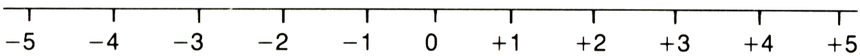


Figure 3 The number line

If a number is found on this line to be to the left of a second number, then the first is less than the second number; if to the right then the first is greater than the second.

Example 1

Test whether the following expressions are true or false for the given values of A and B.

Values A	Values B	Expression
2	5	A>B
2	−5	A>B
−2	−5	A>B
−2	−1	A>B
−5	2	A<B
5	−2	A<B
−3	3	A=B

Solution

To do this we work out the value of each expression using the values given and then use the number line to decide whether or not the assertion is true or false for those particular values. So the solution is:

Values		Assertion		
A	B	Expression	Its value	Its logical state
2	5	A>B	2> 5	F
2	−5	A>B	2>−5	T
−2	−5	A>B	−2>−5	T
−2	−1	A>B	−2>−1	F
−5	2	A<B	−5< 2	T
5	−2	A<B	5<−2	F
−3	3	A=B	−3= 3	F

F = false T = true

SAQ 4
Complete the following table to determine whether the given expressions are true or false for the values given.

Values		Assertion		
A	B	Expression	Its value	Its logical state
3	7	A>B		
5	3	A>B		
−3	5	A>B		
8	5	A<B		
3	9	A<B		
8	−2	A<B		

We are now in a position to use relationships to allow control of a program to jump to a new line when certain conditions are satisfied.

Example 2
In the following program segment after executing line 30, will control pass to line 40 or line 100?

```
10  LET A = −3
20  LET B = 2
30  IF A+B>0 THEN GOTO 100
40
```

Program 5

Solution
−3+2>0 is false, so the branch to 100 will not occur and control will just pass on to line 40.

SAQ 5

In the following program segments after executing line 30, will control pass to line 40 or to line 100?

(a)
```
10 LET A = 7
20 LET B = −8
30 IF A−B<0 THEN GOTO 100
40
```

(b)
```
10 LET X = 3
20 LET Y = −3
30 IF X/Y = −1 THEN GOTO 100
40
```

(c)
```
10 LET P = −1
20 LET Q = 3
30 IF P+Q>Q THEN GOTO 100
40
```

(d)
```
10 LET M = 3
15 LET N = −4
20 LET P = −2
30 IF M−N<N−P THEN GOTO 100
40
```

(e)
```
10 LET R = 1
20 LET S = −2
30 IF R+S>−1 THEN GOTO 100
40
```

Programs 6–10

2.7 FLOWCHARTS

As we have said the principal task for a programmer is to find a suitable way of expressing the solution strategy to solve a particular problem. At this stage we must introduce you to what must be the ugliest word in computer jargon: algorithm. This word is used to mean a general solution strategy, and is defined as a series of instructions or procedural steps for the solution of a specific problem. You will notice that in this case the computer is not mentioned. Apart from that, the definition of program and algorithm are identical. A program then is an algorithm written for a computer.

There are three basic ways of stating an algorithm:

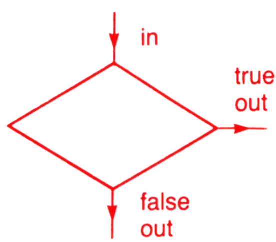
(i) a description
(ii) BASIC coding
(iii) a flowchart

Flowcharts are a bit like blueprints and appeal to those of us who like to see events displayed in pictorial, chart or cartoon form.

We display the different functions within an algorithm by using different shaped boxes.

INPUT AND OUTPUT FUNCTIONS

DECISIONS



ASSIGNMENTS

START/STOP

An → shows the sequence of the algorithm, and the boxes contain appropriate scripts.

The first program from Unit 1 can be expressed in flowchart form as follows:

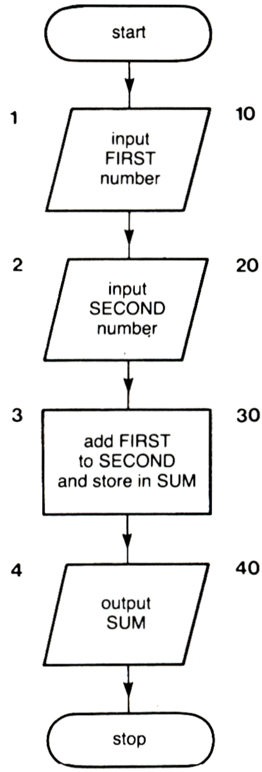


Figure 4 Flowchart of Program 1 from Unit 1

The descriptions of the functions in the boxes are in cryptic-English but could be followed by someone who has no knowledge of BASIC. In that respect we say that we try to keep these descriptions language independent. The numbers on the left-hand side of the boxes refer to the statements in the descriptive algorithm in Figure 4 of Unit 1, and the numbers on the right-hand side of the boxes refer to the statements in the BASIC program in Program 1 of Unit 1.

SAQ 6
Construct a flowchart for the percentage program (Program 1).

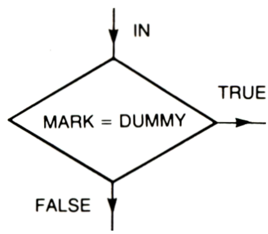
The decision box
You have seen how decisions are effected in BASIC using the IF . . . THEN . . . statement. The logic is:

IF assertion is true THEN go to line X
otherwise (assertion is false) carry on to the next line

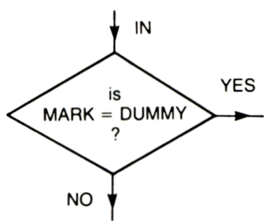
The basic idea of branching to line X or carrying on to the next line is depicted in the flowchart by two exit lines from a decision box. The decision of line 40 of Program 4:

40 IF M = −9999 THEN GOTO 80

could be depicted in language independent form



The assertion MARK = DUMMY may be expressed as a question



Flowchart style is up to you. The test of the effectiveness of a flowchart is whether or not you can follow the flowchart easily some time after its composition! Or, if you are trying to communicate your ideas to somebody else, whether they can follow your flowchart easily.

A flowchart for Program 4 is given below.

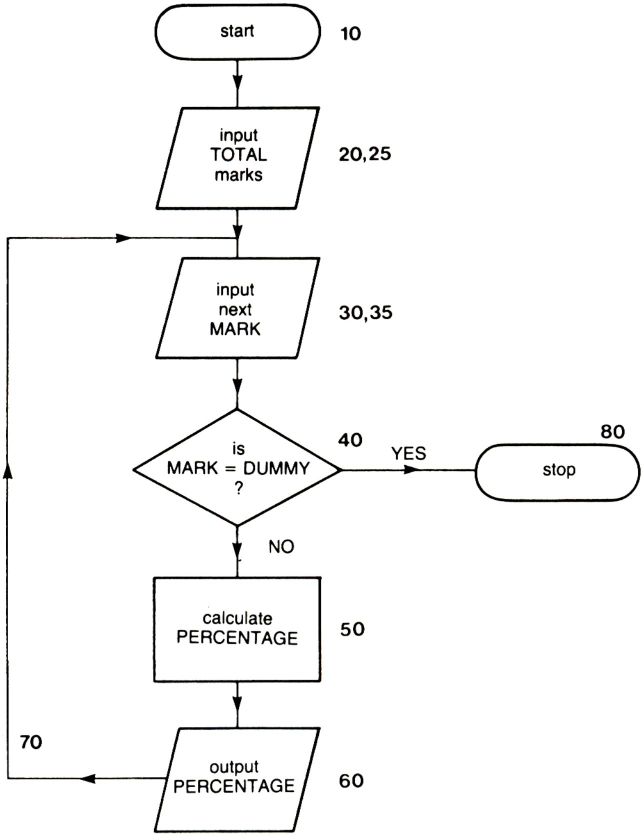


Figure 5 Flowchart for percentage program

The numbers on the right-hand side of the flowchart boxes refer to the statement numbers in the program. Statement 70 GOTO 30 is represented by a loop back to box 30.

SAQ 7
Write a flowchart for the program you wrote in answer to SAQ 3.

Now that we have introduced the idea of flowcharts, we can use them to help plan the structures of the programs that we are going to write.

2.8 Counting

As we have said, computers are good at carrying out lots of repetitive procedures. If however we wish to control these activities rather than just start and stop them,

as we did in the last example, then we must use the computer to count the repetitions for us. If we carry out a specified number of repetitions of an activity, we start counting at the first activity, add one for each subsequent activity, until we reach a predetermined limit. We can depict this procedure in flowchart form.

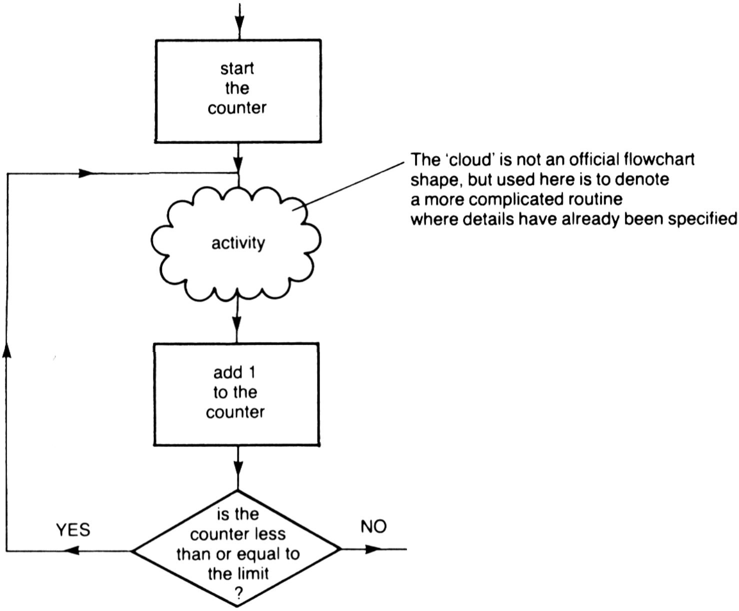


Figure 6 A counter in a flowchart

Notice that there are three parts to the counter:

(i) the procedure that sets the counter to its initial value;
(ii) the procedure for adding 1 to the counter each time the activity is completed;
(iii) the procedure for stopping the counter and leaving the activity when it has been executed the required number of times.

Great care must be taken to ensure that we exit from such a repetitive loop at exactly the point we wish to. As a warning, note that though this loop has counted up to 10, the value of the location COUNT on leaving the loop will be 11. This is a point which you would have to be very careful about if you wished to use the number in COUNT later on in the program.

SAQ 8
How many numbers will be inputted with the flowcharts?

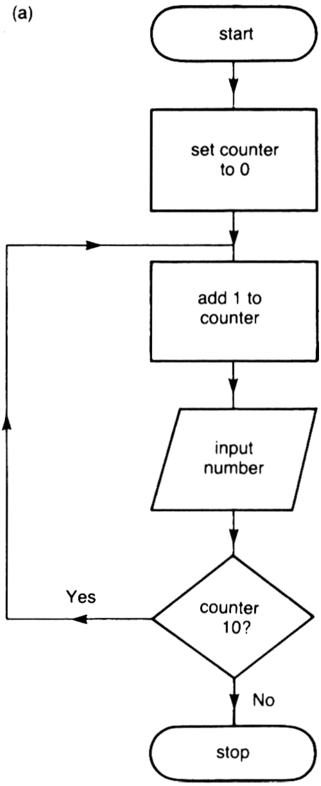


Figure 7a

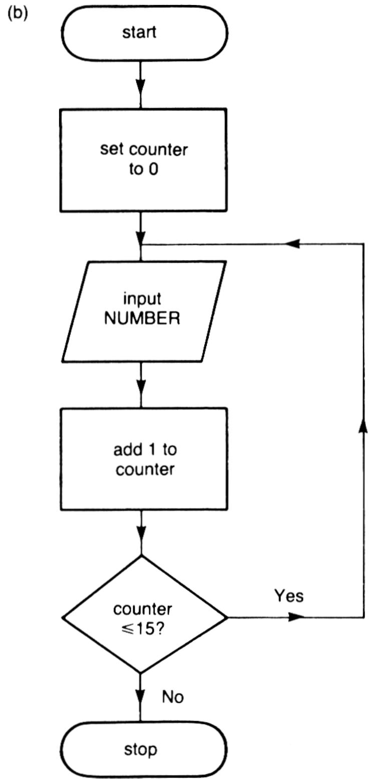


Figure 7b

SAQ 9
Make the following program read 5 numbers by completing the IF . . . THEN GOTO . . . statement.

```
10  LET C=0
20  INPUT N
30  IF C=        THEN GOTO 60
40  LET C = C+1
50  GOTO 20
60  STOP
```

Program 11

Example 3
Write a BASIC program to calculate and output the percentage marks for a group of 5 pupils.

Solution

If we assume that the 'activity' in the cloud in the flowchart of Figure 6 was

input the mark

calculate the percentage

output the percentage

then we can display the algorithm from this solution in flowchart form.

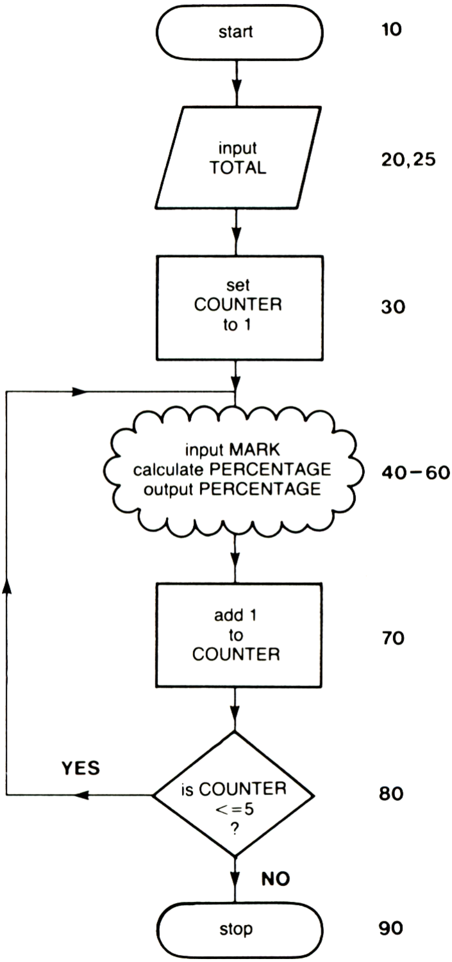


Figure 8 Flowchart for percentage calculation on 5 marks

This tells us the structure of the program that we need to write. The actual program can now be written by modifying Program 4 to incorporate the counter. So the required program is:

```
10  REM★★PERCENTAGES★★
20  PRINT"INPUT THE TOTAL MARKS"
25  INPUT T
26  PRINT T
30  LET C = 1                                  start the counter, 'initialisation'
40  PRINT "INPUT THE NEXT MARK"
45  INPUT M                                    add 1 to the counter,
46  PRINT M                                    'incrementation'
50  LET P = (M/T)★100                          we've added this line to check
60  PRINT"PERCENTAGE",P                        what is happening to the
70  LET C = C+1                                counter at this point in the
75  PRINT"LINE 75","COUNT HERE =",C            program.
80  IF C<=5 THEN GOTO 40                       counting complete?
90  STOP
```

Program 12   Adding a counter to the percentage program

```
RUN
INPUT THE TOTAL MARKS
  75
INPUT THE NEXT MARK
  57
PERCENTAGE         76
LINE 75       COUNT HERE =    2
INPUT THE NEXT MARK
  62
PERCENTAGE         82.6667
LINE 75       COUNT HERE =    3
INPUT THE NEXT MARK
  43
PERCENTAGE         57.3333
LINE 75       COUNT HERE =    4
INPUT THE NEXT MARK
 39
PERCENTAGE 52
LINE 75       COUNT HERE =    5
INPUT THE NEXT MARK
  70
PERCENTAGE         93.3333
LINE 75       COUNT HERE =    6       Note: on emergence from the
                                      loop the value of the counter
                                      is 6.
```

[K] Program 12.

Exercise 1

In question 2 of Assignment 1, you wrote a program to calculate the cost of double glazing a window. If you wanted to use the program to calculate the costs of

several windows, you would have to run the program again and again. This exercise asks you to modify the program to cover more than one window.

Notice that the 'activity' for repetition will be:

input height and width of window calculate cost of installation of this window output the cost

(a) Draw a flowchart algorithm to calculate and output the cost of each of six windows.
(b) Code the algorithm in BASIC. $\boxed{K}$ your answer and try a run for four windows on your microcomputer.
(c) Extend your program to cope with any chosen number of windows, to be specified at the beginning of the program.

Exercise 2
Extend the problem posed in question 1(b) of Assignment 1. The 'activity' for repetition will be:

calculate the yield output the year and its yield calculate the deposit for the next year

(a) Draw a flowchart algorithm to calculate and output the yield for each year up to six years. Check your answer.
(b) Code this algorithm in BASIC in full detail. Check your answer. $\boxed{K}$.
(c) Extend the algorithm to calculate and output the yield for each year up to any chosen number of years, to be specified at the beginning of the program. Check your answer. $\boxed{K}$.

2.9 Comparisons

We have seen how the BASIC language allows us to compare two numbers. We often wish to decide whether a particular value is larger or smaller than another. This is a process which is fundamental to sorting items of data. Throughout the course we will consider sorting methods in some detail, so let's start with the simplest case.

Example 4
Devise an algorithm in descriptive form to input two numbers and to output the larger of the two.

Comment
We have expressed our algorithms as flowcharts throughout most of this unit so this time we will use the decriptive method introduced in Unit 1.

Solution

1. Start.
2. Input first number.
3. Input second number.
4. If first number > second number then go to 7 otherwise carry on to 5.
5. Output second number.
6. Go to 8.
7. Output first number.
8. Stop.

This is not the neatest solution, but is close to the layman's 'first attempt' at the problem. We will seek neater solutions when we return to sorting methods in a later unit.

Assignment 2

1. Devise a flowchart and write a BASIC program to input two numbers and output the smaller of the two. Modify the program so that it will process (a) five pairs of numbers, (b) any number of pairs of numbers.

2. Extend the 'mark – percentage' algorithm on page 41, and express it in the form of flowchart and BASIC program:

(a) to accommodate a class of any size;
(b) to calculate the average percentage mark;
(c) to pick out the highest mark.

Objectives of Unit 2

Now that you have completed this Unit, check that you are able to:

Combine literal printing and variable print in PRINT statements ☐

Use , to space PRINT statements ☐

Use GOTO to repeat the use of a program ☐

Use a dummy value to terminate a program ☐

Use IF . . . THEN GOTO . . . ☐

Find the logical state of assertions including >, <, = ☐

Construct flowcharts ☐

Insert counters in flowcharts and programs to control the repeated use of part of a program or flowchart ☐

Answers to SAQ's and Exercises

SAQ 1

(a) AREA48

(b) LENGTH8 WIDTH6

(c)
```
LENGTH          WIDTH
8               6
```

(d) PRINT "LENGTH", B,

(e)
```
PRINT "LENGTH", B
PRINT "WIDTH", C
PRINT "AREA", A
```

(f) PRINT "LENGTH", "WIDTH"

SAQ 2

All you need to do is add

```
35 GOTO 10
```

SAQ 3

```
10 INPUT N
15 IF N = -9999 THEN GOTO 40
20 LET S = N★N
30 PRINT S
35 GOTO 10
40 STOP
```

Program 13

(Note that this terminating value is not quite as satisfactory as using −9999 as a dummy mark. You can't have a mark of −9999, but you might perhaps want to square −9999; this program would refuse to do it.)

SAQ 4

Values		Assertion		
A	B	Expressions	Its value	Its logical state
3	7	$A>B$	$3>7$	F
5	3	$A>B$	$5>3$	T
−3	5	$A>B$	$-3>5$	F
8	5	$A<B$	$8<5$	F
3	9	$A<B$	$3<9$	T
8	−2	$A<B$	$8<-2$	F

If you got any of these wrong, look at them again on the number line

```
_____A_____________B_____     A to the left of B means A<B true
_____B_____________A_____     A to the right of B means A>B true
```

SAQ 5

(a)	40	(b)	100	(c)	40
(d)	40	(e)	40		

Your computer could help solve this problem for you. The statements

```
    40  PRINT "40"
and 100  PRINT "100"
```

will cause the appropriate line to be output.

The following programs show how you could have solved (a) and (b) above.

Program to solve (a)

```
10  LET A = 7
20  LET B = -8
30  IF A-B< 0 THEN GOTO 100
40  PRINT "40"
50  GOTO 999
100 PRINT "100"
999 STOP
```

Program 14

Program to solve (b)

```
10  LET X = 3
20  LET Y = -3
30  IF X/Y=1 THEN GOTO 100
40  PRINT "40"
50  GOTO 999
100 PRINT "100"
999 STOP
```

Program 15

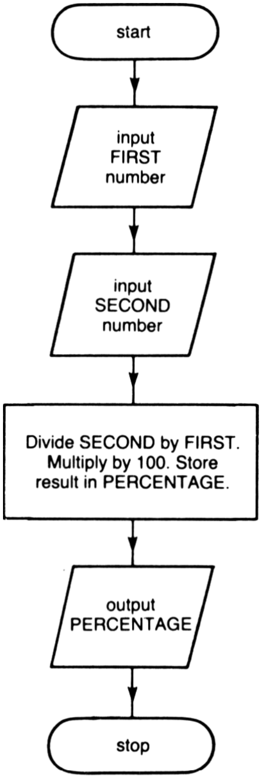
Effect of running Program 14

```
40
9/999
```

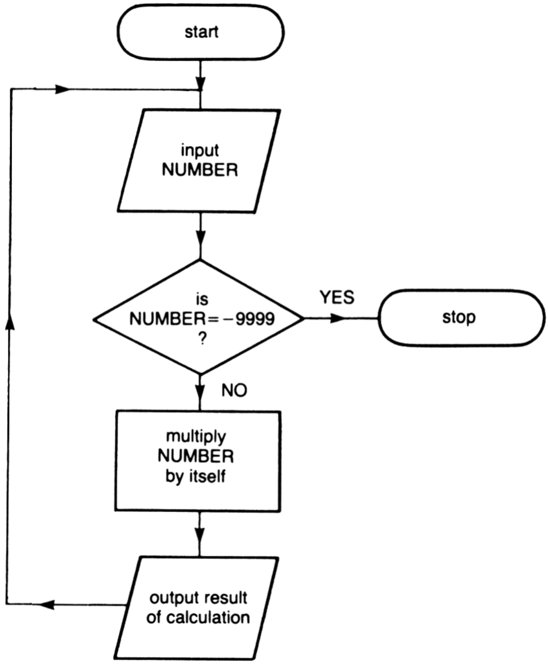
Effect of running Program 15

```
100
9/999
```

SAQ 6



SAQ 7



SAQ 8

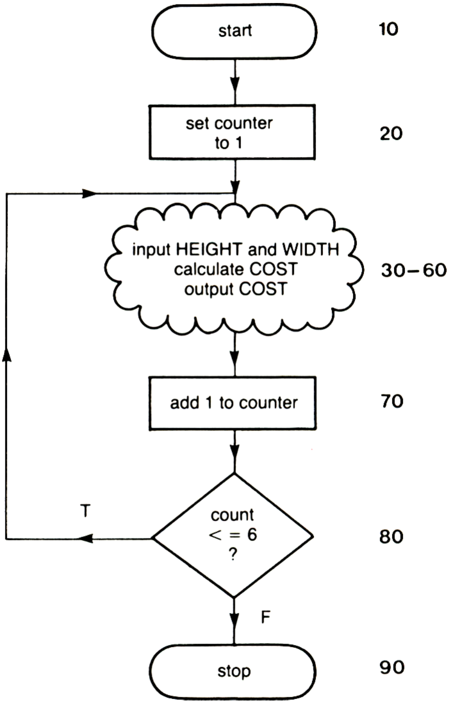
(a) 10 (b) 16

SAQ 9

30 IF C = 4 THEN GOTO 60

Exercise 1

1(a)



1(b)

```
10  REM★★COST OF DOUBLE GLAZING★★
20  LET C = 1
30  PRINT "ENTER HEIGHT IN METRES"
35  INPUT H
36  PRINT H
40  PRINT"ENTER WIDTH IN METRES"
45  INPUT W
46  PRINT W
50  LET K=14★(H+W)+40★(H★W)+50
60  PRINT "WINDOW"; C; "COST"; K
70  LET C = C+1
80  IF C<=6 THEN GOTO 30
90  STOP
```

Lines 20, 70 and 80 are the counter

Program 16

```
RUN
ENTER HEIGHT IN METRES
  1.5
ENTER WIDTH IN METRES
  2
WINDOW 1 COST 219
ENTER HEIGHT IN METRES
```

```
   1.5
ENTER WIDTH IN METRES
   3
WINDOW 2 COST 293
ENTER HEIGHT IN METRES
   1.5
ENTER WIDTH IN METRES
   4
WINDOW 3 COST 367
ENTER HEIGHT IN METRES
   2.5
ENTER WIDTH IN METRES
   2
WINDOW 4 COST 313
etc:
```

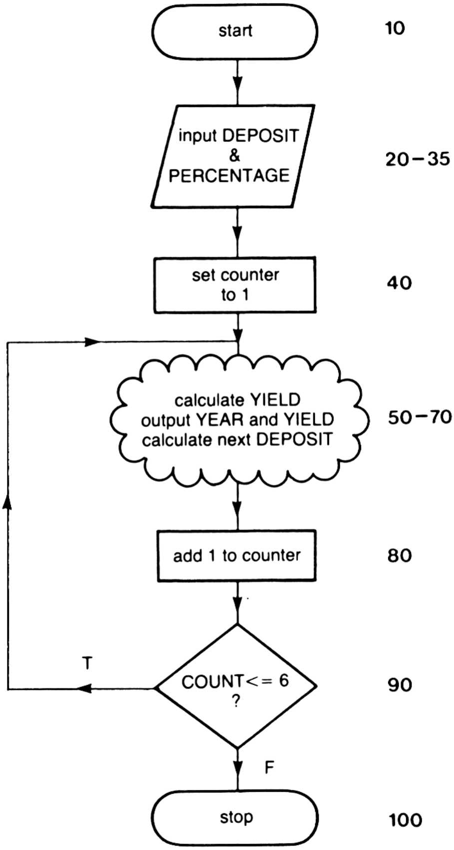
1(c)

```
10  REM★★COST OF DOUBLE GLAZING★★
12  PRINT"ENTER NUMBER OF WINDOWS";
14  INPUT N
15  PRINT N
20  LET C = 1
30  PRINT"ENTER HEIGHT IN METRES";
35  INPUT H
36  PRINT H
40  PRINT"ENTER WIDTH IN METRES";
45  INPUT W
46  PRINT W
50  LET K = 14★(H+W)+40★(H★W)+50
60  PRINT"WINDOW ";C;" COST ";K
70  LET C = C+1
80  IF C <=N THEN GOTO 30
90  STOP
```

Program 17

Exercise 2

2(a)



2 (b) Program

```
10  REM★★COMPOUND INTEREST★★
20  PRINT "ENTER DEPOSIT  ";
25  INPUT D
26  PRINT D
30  PRINT"ENTER PERCENTAGE INTEREST ";
35  INPUT P
36  PRINT P
37  PRINT———————————————— separates table from above input lines
40  LET C = 1
50  LET Y = (P★D)/100
60  PRINT "YEAR ";C;"____YIELD____";Y
70  LET D = D+Y
80  LET C = C+1
```

```
90  IF C <= 6 THEN GOTO 50
100 STOP
```

Program 18

```
RUN
ENTER DEPOSIT  100
ENTER PERCENTAGE INTEREST  15
YEAR  1            YIELD          15
YEAR  2            YIELD          17.25
YEAR  3            YIELD          19.8375
YEAR  4            YIELD          22.813125
YEAR  5            YIELD          26.235094
YEAR  6            YIELD          30.170358
READY
```

2(c) As above, except:

```
15  PRINT"ENTER NUMBER OF YEARS ";
16  INPUT N
17  PRINT N
90  IF C <= N THEN GOTO 50
```

Program 19

UNIT 3
Strings

3.1	What is a string?	64
3.2	More about strings	66
3.3	PRINT . . .;. . .	66
3.4	INPUT ". . .";. . .	69
3.5	Numbers and strings in PRINT statements	70
3.6	Standard letters	73
3.7	Files and Records	75
3.8	Sorting	78
	Assignment 3	82
	Objectives of Unit 3	82
	Answers to SAQ's and Exercises	83
	Appendix	87

3.1 What is a string?

The first two units were concerned with processing numbers. The layman often sees the computer as a 'number cruncher' but this is certainly not the main function of a computer, especially in a commercial environment. In this Unit we will see how a computer may be used to manipulate characters using the BASIC language.

By character we mean the alphabet in capitals, the ten digits 0–9, punctuation marks and some special characters as follows:

@, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, [, \,], ↑, ←,

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, <, =, >, ?, Space, !, ,, #, $, %, &, ., (,), +, ★, −, /.

You have learnt to write programs using numbers (3, 57, −92, etc.) and variables (A, X, Z, etc.). BASIC also allows us to enter characters into the computer in groups.

These groups of characters are referred to as strings. Some examples of strings are:

CAT	(a word)
MARGARET THATCHER	(a name)
Z9)?27	(a mixture of characters)
ABC 123W	(a car registration number)

i.e. a string can be any mixture of characters – even a space is a very important character in a string!

As far as BASIC is concerned, a number is treated as a number when it is to be used to do some arithmetic, otherwise it is considered to be a string of numeric characters. When we look at a car or telephone number we see it as a group of numeric characters; we would not use this collection of digits to do any serious arithmetic.

Store locations for strings

How then can we signal to the computer that the group of characters which we are entering should be treated as numbers for arithmetic purposes, or as just a string of characters? The distinction is made in BASIC by how we label the storage locations into which we put the characters. If a store location name is followed by the symbol $ then the characters which are entered into that location are treated as strings of characters.

You saw in Unit 1 that the store locations for numbers in ZX81 BASIC are the 286 locations:

A, A0, . . . A9
B, B0, etc

The store locations for strings in a minimal BASIC are the 26 locations:

A$, B$, C$, . . . Z$.

You read these out loud as follows:

A$	A string	or	A dollar
B$	B string	or	B dollar

Thus you can now think of a microcomputer as having two areas for store locations: one for numbers and one for characters. This is illustrated in Figure 1.

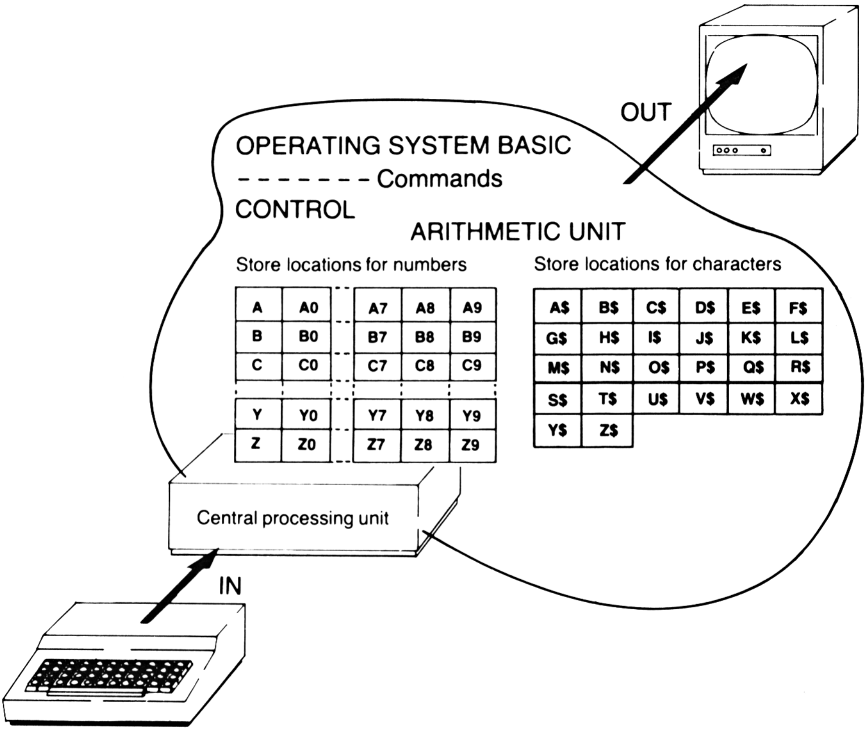


Figure 1 A summary of our system so far

" " with strings

Usually we have to show the computer that we want our string of characters to be treated as a string. To do this we put " " around the string. Thus we write

```
10  LET Q$ = "HELLO"
```
and
```
30  IF Q$="HELLO" THEN GOTO 80
```
and so on.

SAQ 1

Which of the following are valid store location names for strings:

(a) A$ (b) M8 (c) T7$ (d) B9 (e) C$3 (f) 8P$ (g) 2$ (h) K$

SAQ 2
Which of the following are correct BASIC statements:

(a) LET A = 87
(b) LET B$ = "FRED"
(c) LET M$ = 9583
(d) LET K8 = "JAM POT"
(e) LET L17 = 38

3.2. More about strings

'How long is a piece of string?' is a pertinent question here. In other words, how many characters can be input, stored or output as a single group? The ZX81 allows strings to be of any length within the currently available memory capacity of the computer. But initially in this course we will assume that a store location for strings will hold up to 40 characters, and that this restriction will apply to inputting and outputting strings. So you must now think of a string memory location, not just as a labelled pigeon-hole, but as a location with 40 sub-divisions, as shown in Figure 2.

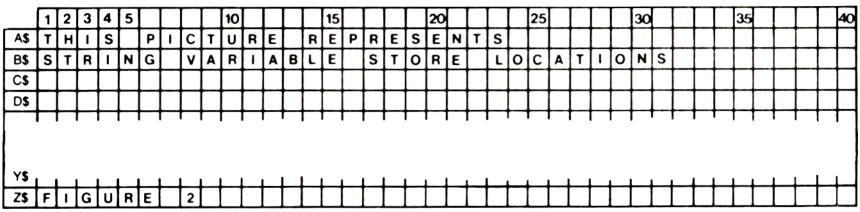


Figure 2 Size of string store locations

We have been discussing strings as if they were new but you have met them before in Unit 1. There we used the PRINT statement to output messages which were enclosed in quotation marks. We implied that the string enclosed in quotation marks was output literally character by character. Then we saw in Unit 2 how the commas between the elements of a PRINT statement caused the strings to be spaced out across the screen or printer.

3.3 PRINT . . .;. . .

From what we have covered so far you will realise that the layout of information on the screen is very important. This is just as true for strings as it is for numbers.

When handling textual information, ie strings of characters in the form of words or codes, we want the strings to be printed as in a sentence and not spaced out across the screen in print zones. The PRINT . . .;. . . statement achieves this effect for us. PRINT H$;T$ will take the characters in store location H$ and print them on

the left-hand side of the output device followed immediately by the characters from location T$.

In the next few pages we are going to simulate a data recording service of the not too distant future and use it to demonstrate the inputting and outputting of strings. Let's start by writing a program which simulates a telephone answering service.

```
10   REM★★TELE ANSWER★★
30   PRINT "HELLO"
40   PRINT "PLEASE STATE YOUR PHONE NUMBER"
50   INPUT T$
51   PRINT T$
60   PRINT ——————————————— this prints a blank line
70   PRINT "HELLO", T$
80   PRINT
90   PRINT "HELLO"; T$
100  PRINT
110  PRINT "HELLO "; T$
120  PRINT
130  PRINT "HELLO"; " "; T$
140  STOP
```

Program 1 Printing strings

To help you analyse this program we have put below a 'trace' at the side of a typical run. The trace indicates which line in the program generates which line in the output.

```
                                                      Trace
RUN
HELLO                                                 ... 30
PLEASE STATE YOUR TELEPHONE NUMBER                    ... 40
 58632                                                ... 50
"L"                                                   ... 60
HELLO                58632                            ... 70
                                                      ... 80
HELLO58632                                            ... 90
                                                      ... 100
HELLO 58632                                           ... 110
                                                      ... 120
HELLO 58632                                           ... 130
```

Comments

Trace 50 INPUT T$ generated "L". Our response was 58632 which the computer treats as a string and not as a number. (If we had written INPUT T, then 58632 would be treated as a number.)

Trace 70 The PRINT . . .,. . . on line 70 prints the 5 of the telephone number at the 16th print position across the output line, whereas

Trace 90 The PRINT . . .;. . . of line 90 prints the 5 immediately adjacent to the O of HELLO.

Neither way is satisfactory, but:

Trace 110 Trace 130 Lines 110 and 130 show alternative ways of introducing the required spaces, either by printing HELLO▽(110) or by inserting the string ▽ in its own right into the output statement. (▽ indicates a space.)

[K] Program 1.

SAQ 3
Study this program and work out what the print output will be. Write down the output in the grid below.

```
 10 PRINT "PRINT LAYOUT"
 20 LET B$ = "BASIC"
 30 LET C$ = "COURSE"
 40 PRINT
 50 PRINT B$, C$
 60 PRINT
 70 PRINT B$; C$
 80 PRINT
 90 PRINT B$; "  "; C$
100 STOP
```

Program 2

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1

SAQ 4
Write a program which would print out the following:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
B	A	S	I	C												B	A	S	I	C
B	A	S	I	C	B	A	S	I	C	B	A	S	I	C	B	A	S	I	C	
B	A	S	I	C		B	A	S	I	C		B	A	S	I	C				

3.4 INPUT "...";...

We have used the PRINT"... " as a prompt for an INPUT statement in several of our programs so far. Most BASIC's have a facility to allow us to combine these two into one statement. Thus in the above program, we could replace:

```
40  PRINT "PLEASE STATE YOUR TELEPHONE NUMBER"
50  INPUT T$
```

with

```
40  INPUT "PLEASE STATE YOUR TELEPHONE NUMBER";T$
```

This facility is not available on the ZX81 but the same result can be achieved using the three lines:

```
40  PRINT "PLEASE STATE YOUR TELEPHONE NUMBER?";
41  INPUT T$
42  PRINT T$
```

Note the ';' at the end of line 40 which ensures that line 42 prints T$ after PLEASE STATE YOUR TELEPHONE NUMBER?

The next program demonstrates various PRINT effects. Since we need to use "HELLO" several times we first store it in location H$ at the start of the program.

```
10  REM★★TELE ANSWER★★
30  LET H$="HELLO"
40  PRINT H$
50  PRINT "WHAT IS YOUR PHONE NUMBER?";
51  INPUT T$
52  PRINT T$
60  PRINT
70  PRINT H$, T$
80  PRINT
90  PRINT H$; T$
100 PRINT
110 PRINT H$;"  ";T$
120 STOP
```

Program 3 The computer asks the questions

```
RUN
HELLO
WHAT IS YOUR TELEPHONE NUMBER? 58632

HELLO           58632

HELLO58632

HELLO 58632
```

[K] Program 3.

SAQ 5

What would appear on the screen when this program was run assuming your name is John Smith and your age 45?

```
10  LET T$="THANK YOU"
20  PRINT "WHAT IS YOUR NAME? ";
30  INPUT N$
40  PRINT N$
50  PRINT
60  PRINT "WHAT IS YOUR AGE? ";
70  INPUT A$
80  PRINT A$
90  PRINT
100 PRINT
110 PRINT T$;"     ";N$;"      ";A$
120 STOP
```

Program 4

3.5. Numbers and strings in print statements

We could have entered the telephone number of the previous program into a numeric store location. We would of course soon run into problems if the number were too long, or contained spaces (eg 01 693 4539). Let's compare how BASIC would output this data from numeric and string store locations.

In this program note how we use the string of characters in S$ to print a scale across the output page.

```
10  REM★★TELE ANSWER★★
30  LET H$="HELLO"
35  LET S$="1234567890123456789012345"
40  PRINT H$
50  PRINT "WHAT IS YOUR PHONE NUMBER? ";
51  INPUT T$
52  PRINT T$
55  PRINT "PLEASE TYPE IT IN AGAIN? ";
56  INPUT T
57  PRINT T
58  PRINT
60  PRINT S$
70  PRINT H$, T$
75  PRINT H$,T
80  PRINT S$
90  PRINT H$;T$
95  PRINT H$;T
100 PRINT S$
110 PRINT H$;" ";T$
115 PRINT H$;" ";T
120 STOP
```

Program 5   Printing strings and numbers

```
RUN                                                Trace
HELLO
WHAT IS YOUR TELEPHONE NUMBER? 58632
PLEASE TYPE IT IN AGAIN? 58632
1234567890123456789012345       60 S$ numbers each print
HELLO          58632               position across the page.
HELLO          58632            75 note that the first digit 5 is
1234567890123456789012345          placed at the 16th position.
HELLO58632                         the 15th is reserved for the
HELLO58632                      95
1234567890123456789012345
HELLO 58632
HELLO 58632                     115

RUN
HELLO
WHAT IS YOUR TELEPHONE NUMBER?–58632
PLEASE TYPE IT IN AGAIN? – 58632
1234567890123456789012345
HELLO         –58632
HELLO         –58632            75
1234567890123456789012345
HELLO–58632                        note the effect when T$
HELLO–58632                     95 and T both hold –58632.
1234567890123456789012345
HELLO –58632
HELLO –58632                    115
```

[K] Program 5.

SAQ 6

Write a program to input your name as a string and your age as a number and to output the message, 'My name is   and I am   years old' with the normal spacing.

Data recording service

The following is a further example of how print layout is achieved in BASIC. We can imagine that in the not too distant future our TV set, telephone and computer will be linked together as an 'intelligent' terminal. On seeing an attractive advertisement we may 'dial' a number and the following dialogue might ensue.

```
                                                   Trace
HELLO                                               30
THIS IS A DATA–RECORDING SERVICE                    40
                                                    50
PLEASE ENTER THE DETAILS AS REQUESTED               60
                                                    70
YOUR NAME? C. A. SMITH                              80
YOUR TELEPHONE NUMBER? 23685                        90
NUMBER OR NAME OF HOUSE? 77                         100
ROAD? CHALMERS ROAD                                 110
```

```
TOWN OR CITY? WORTHING                                     120
YOUR POSTAL CODE? BR7 9QY                                  130
                                                           140
                                                           150
                                                           160
THANK YOU FOR ENQUIRY                                      170
                                                           180
YOUR PERSONAL DETAILS HAVE BEEN RECORDED AS:               190
NAME C. A. SMITH TELEPHONE NO. 23685                       200
ADDRESS 77 CHALMERS ROAD                                   210
            WORTHING BR7 9QY                               220
                                                           230
DETAILS OF OUR SERVICES AND PRODUCTS                       240
WILL BE SENT TO YOU                                        250
YOUR PERSONAL DETAILS WILL REMAIN CONFIDENTIAL             260
```

(Of course, instead of 'will be sent to you' it will eventually be 'will now be output to your terminal', and then the only details needed to be input would be a subscriber code.)

This simulated dialogue was achieved by the following program.

```
10  REM★★THIS IS A DATA RECORDING SERVICE★★
30  PRINT "HELLO . . ."
35  PRINT
40  PRINT "THIS IS A DATA RECORDING SERVICE"
50  PRINT
60  PRINT "PLEASE ENTER DATA AS REQUESTED"
70  PRINT
80  PRINT "YOUR NAME? ";
81  INPUT N$
82  PRINT N$
90  PRINT "YOUR TELEPHONE NUMBER? ";
91  INPUT T$
92  PRINT T$
100 PRINT "NUMBER OR NAME OF HOUSE? ";
101 INPUT H$
102 PRINT H$
110 PRINT "ROAD? ";
111 INPUT R$
112 PRINT R$
120 PRINT "TOWN OR CITY? ";
121 INPUT C$
122 PRINT C$
130 PRINT "YOUR POSTAL CODE? ";
131 INPUT P$
132 PRINT P$
140 PRINT
150 PRINT
160 PRINT
170 PRINT "THANK YOU FOR YOUR ENQUIRY"
180 PRINT
```

```
181 PRINT "................................"
182 PRINT
190 PRINT "YOUR PERSONAL DETAILS HAVE BEEN RECORDED AS:"
195 SCROLL
196 SCROLL
200 PRINT "NAME ";N$
201 SCROLL
204 PRINT "TELEPHONE NUMBER "; T$
205 SCROLL
210 PRINT "ADDRESS ";H$;" ";R$
215 SCROLL
220 PRINT "          ";C$;" ";P$
230 SCROLL
235 SCROLL
240 PRINT "DETAILS OF OUR SERVICES AND"
241 SCROLL
242 PRINT "PRODUCTS WILL BE SENT TO YOU"
245 SCROLL
255 SCROLL
260 PRINT "YOUR PERSONAL DETAILS"
265 SCROLL
270 PRINT "WILL REMAIN CONFIDENTIAL"
275 STOP
```

Personal details scroll up the page as if being typed.

Program 6   Data recording service

[K] Program 6.

3.6 Standard letters

A data recording service, such as we have just looked at, may be in the future, but standard personalised letters are with us now. Such a letter would be composed on a word-processor, but if your micro does not have word processing facilities available, you could achieve modest results using BASIC. Your own choice of letter will be left to you in Exercise 2.

Example 1
A bank recruiting office receives many enquiries about employment. Its policy is to interview suitable applicants initially at its local branch. A stereotyped letter is sent from the recruiting office to each applicant containing individual details of the proposed interview. Devise a BASIC program to write such a letter.

Solution
The following program would do this job.

```
10  REM★★LETTER WRITER★★
30  INPUT A$ ———— applicant's name
40  INPUT B$ ———— date of letter of application
50  INPUT C$ ———— name of interviewer
60  INPUT D$ ———— time of interview
```

```
70  INPUT E$   ———————————————— date of interview
80  INPUT F$   ———————————————— location of interview
90  INPUT G$   ———————————————— name of employee replying
100 PRINT
110 PRINT
120 PRINT"DEAR ";A$;","
130 PRINT
140 PRINT "THANK YOU FOR YOUR LETTER OF"
145 PRINT B$;"."
150 PRINT "WE INVITE YOU TO ATTEND FOR"
155 PRINT "INTERVIEW WITH ";C$;" AT"
160 PRINT D$; " ON "; E$
170 PRINT "AT OUR "; F$; " BRANCH"
180 PRINT
190 PRINT "YOURS SINCERELY,"
200 PRINT
210 PRINT
220 PRINT
230 PRINT G$
240 PRINT
250 PRINT
260 STOP
```

Program 7 Bank interview letter

This would result in the following run:

```
                                                        Trace
RUN
? MISS JONES                                            30
? 13TH OCTOBER                                          40
? MR FELLOWS                                            50
? 10.00 AM                                              60
? 20TH OCTOBER                                          70
? HIGH ST. SIDCUP                                       80
? C. A. SIDWELL                                         90
                                                        100
                                                        110
DEAR MISS JONES,                                        120
                                                        130
THANK YOU FOR YOUR LETTER OF 13TH OCTOBER.              140
WE INVITE YOU TO ATTEND FOR INTERVIEW WITH              150
MR FELLOWS AT 10.00 AM ON 20TH OCTOBER                  160
AT OUR HIGH ST. SIDCUP BRANCH.                          170
                                                        180
YOURS SINCERELY,                                        190
                                                        200
                                                        210
                                                        220
C.A. SIDWELL                                            230
```

The user of the program might find it difficult to use since all he gets is a series of prompts $\boxed{\text{L}}$. He might, therefore, make a skeletal aide-memoire to remind him of the structure of the letter.

```
A$
B$
C$
D$
E$
F$
G$

DEAR A$,

THANK YOU FOR YOUR LETTER OF B$.
WE INVITE YOU TO ATTEND FOR INTERVIEW WITH
C$ AT D$ ON E$
AT OUR F$ BRANCH.

YOURS SINCERELY,

G$
```

[K] Program 7

Exercise 1
An estate agent periodically sends out a letter to check whether clients on his books are still looking for a property, and that his details of their requirements (eg type of property, price range, etc) are correct. Devise a BASIC program to write such a letter.

Exercise 2
We all write letters requesting things, eg details of a product, a service, a holiday, a job, etc. Devise a BASIC program to write a letter which will cover as wide a range of applications as possible, leaving you to fill in only the particular details of each enquiry.

3.7 Files and records

Quite often we want to record data which is of one kind, i.e. it makes up a file of information. A telephone directory is a good example of what in data processing is called a file, that is a collection of similar records. Each record has the form

Name	Address	Telephone number

and is said to consist of a number of fields – in this case three: name, address and telephone number. A record is then a collection of fields and a file is a collection of records. A telephone directory is arranged in alphabetical order of surnames which gives it a simple structure.

Comparing strings
We may wish to compare strings. Suppose, for example, we have a personal telephone directory in our microcomputer and we wish to find out whether SMITH

is in our record. The computer will have to compare the string "SMITH" against all the strings in the name field of our directory. It can do this very easily because each letter is represented inside the computer by a binary code. Thus

A is 100 0001

B is 100 0010

and so on. (See the Appendix to this unit for a full list of binary codes.) So words placed in alphabetical order on paper will be represented in the computer by codes in numerical order.

Thus if

A$ = CAT
B$ = DOG
C$ = CAT
D$ = FISH
E$ = CATS

A$ = C$

But B$ > A$ (it is further on in the alphabet)
and E$ > A$ (the extra S on CAT puts it after CAT in alphabetical order.)

We shall now use this facility in our examples.

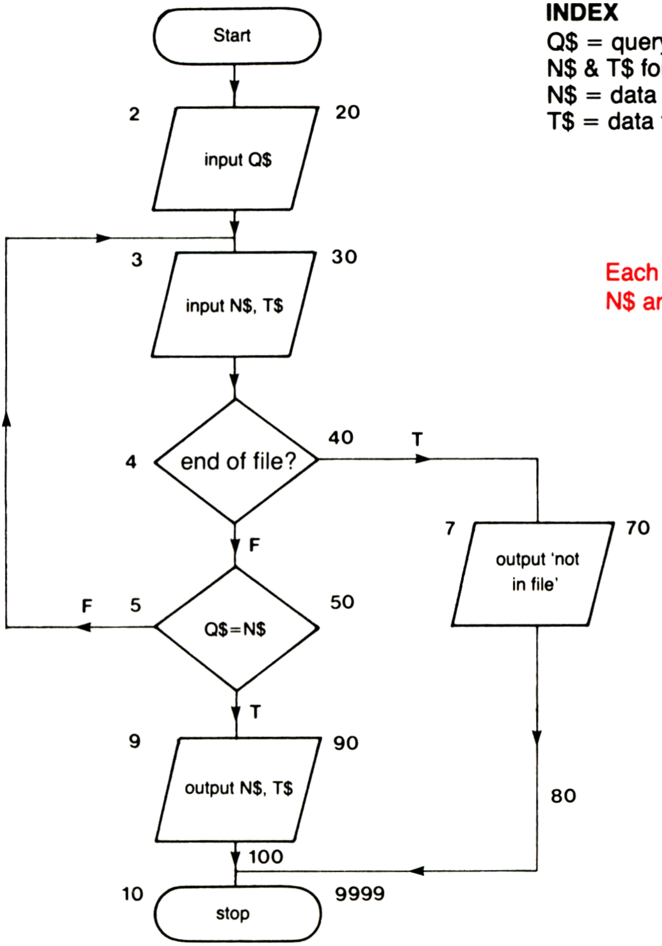
Example 2
Set up a list of names and associated telephone numbers. Write a BASIC program to search through the file to find a particular name, and if found then output the associated telephone number.

Solution
We could attempt a descriptive algorithm as follows:

1. Start.
2. Input query name.
3. Read next record of the data file (i.e. name and number).
4. If the end of the file has been reached then output message 'not found in file' and go to 7 otherwise carry on to 5.
5. If query name = data name then output name and number and go to 7 otherwise carry on to 6.
6. Return to 3 for next record.
7. Stop.

However, BASIC does not generally allow statements as complicated as 4 and 5, and so we have to split up these statements as shown in the next algorithm:

1. Start.
2. Input query name.
3. Read next record from data file.
4. If the end of file has been reached then go to 7 otherwise carry on to 5.
5. If query name = data name then go to 9 otherwise carry on to 6.
6. Return to 3 for next record.
7. Output message 'not in file'.
8. Stop.
9. Output name and number.
10. Stop.



INDEX
Q$ = query name
N$ & T$ form data record
N$ = data name
T$ = data telephone number

Each file has two fields: N$ and T$.

Figure 3 Searching a telephone directory

Now each record contains two fields on this occasion: name and number.

	Field 1	Field 2
	Name N$	Telephone Number T$
e.g.	BENNY	1234

so each item must contain information for each field. At this stage in the course we have only very limited means of storing the data. The first pair

BENNY 1234

is stored as

```
30  LET N$="BENNY"
32  LET T$="1234"
```

The second pair as

```
40  LET N$="COPPER"
42  LET T$="9823"
```

etc. This is a little clumsy but at least we can search the list to find the line at which Q$=N$.

```
10  REM★★TELEPHONE DIRECTORY★★
20  INPUT "SURNAME OF PERSON SOUGHT"
21  INPUT Q$
22  PRINT Q$
25  REM★★DATA★★
30  LET N$="BENNY"
32  LET T$="1234"
35  IF Q$=N$ THEN GOTO 200
40  LET N$="COPPER"
42  LET T$="9823"
45  IF Q$=N$ THEN GOTO 200
50  LET N$="DRAPER"
52  LET T$="1850"
55  IF Q$=N$ THEN GOTO 200
190 PRINT Q$; " IS NOT IN FILE"
195 STOP
200 PRINT "NUMBER FOR "; Q$;" IS "; T$
210 STOP
```

space for more names and telephone numbers

Program 8   Telephone directory

```
SURNAME OF PERSON SOUGHT?
BENNY
NUMBER FOR BENNY IS 1234

SURNAME OF PERSON SOUGHT?
FRED
FRED IS NOT IN FILE
```

K Program 8

SAQ 10
What changes would you have to make to Program 8 to input a person's telephone number and output the subscriber's name or 'is not in file'.

3.8 Sorting

You will have noticed that the telephone directory data in Example 2 was in alphabetical order as you would expect. It would be difficult for the user in normal

practice if this were not so. However, our solution to this searching problem did not use this information; we just searched through the data file record by record until we found the name, or reached the end of the file. Our algorithm would have worked equally well had the data not been in alphabetical order. We shall spend some time later in the course sorting and searching data, at which point you will realise the advantages of sorting data into alphabetical order.

Let's make a modest start with this problem.

Example 3
Write a BASIC program to enter two names into the computer and output that name which would come first in alphabetical order.

Solution
Descriptive algorithm

1. Start.
2. Input first name.
3. Input second name.
4. If first name < second name then 7 otherwise carry on to 5.
5. Output second name.
6. Go to 8.
7. Output first name.
8. Stop.

An outline flowchart for the solution of this problem is shown in Figure 4.

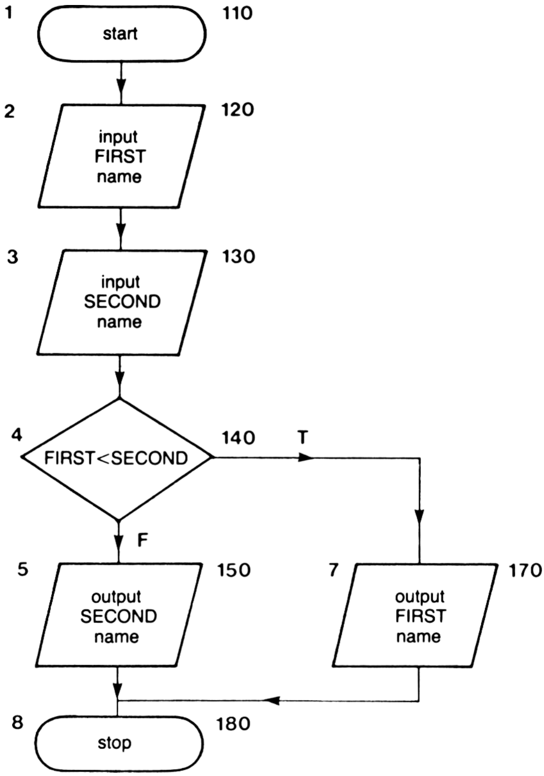


Figure 4 Finding the first of two in alphabetical order

```
110 REM★★FIRST IN ALPHA-ORDER★★
120 PRINT "FIRST NAME? ";
121 INPUT A$
122 PRINT A$
130 PRINT"SECOND NAME? ";
131 INPUT B$
132 PRINT B$
140 IF A$<B$ THEN GOTO 170
150 PRINT "FIRST IN ALPHA-ORDER IS  ";B$
160 GOTO 180
170 PRINT "FIRST IN ALPHA-ORDER IS  ";A$
180 STOP
```

Program 9

Typical run

RUN
FIRST NAME? BROWN

SECOND NAME? SMITH
FIRST IN ALPHA-ORDER IS BROWN
RUN
FIRST NAME? SMITH
SECOND NAME? BROWN
FIRST IN ALPHA-ORDER IS BROWN

K Program 9.

The 3 card trick
Suppose now that we wanted to input three names and output the name that comes first in alphabetical order. A standard solution to this would be to follow the approach in Figure 5.

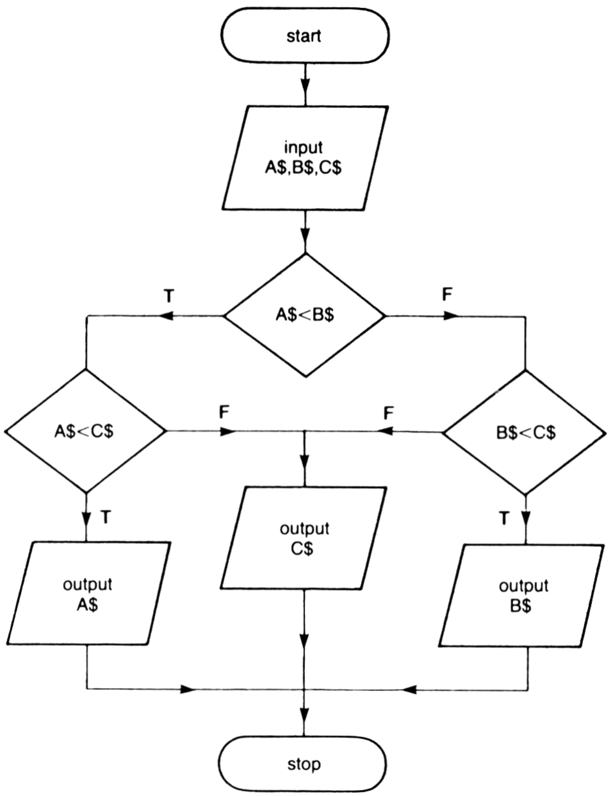


Figure 5 Finding the first of three in alphabetical order

When asked to solve this problem most students present an answer similar to the algorithm in Figure 5. It is a perfectly good solution, but it bodes ill for the future.

Future trouble stems from the fact that we have had to utilise 3 decision and 3 output functions. This method would overtax our patience and ingenuity if we tried to repeat it for 4, 5 . . . let alone 10 names. The fundamental problem is allowing each variable to retain its own individual storage location, and how we are best able to label that location.

A simpler method of solving this problem is to input the names one by one and to store lowest-so-far in A$. The program retains only lowest-so-far, destroying all the other discarded data. In the next Exercise we suggest you try this approach to solving the problem.

Exercise 3
Write a BASIC program to input three names and output that name which would come first in alphabetical order using the method discussed in the last few lines.

Assignment 3

1. Compose a descriptive algorithm and draw a flowchart to accompany the BASIC program in program 15.
2. Modify your program for Exercise 4 to count the numbers of correct and incorrect responses, and to give a summary of the marks at the end of the quiz.
3. Devise an algorithm and write a BASIC program to do the following task. Input several words as individual letters and count the number of vowels and consonants contained in the words, and output the totals of these counts together with the ratio total number of vowels/total number of consonants.

Objectives of Unit 3

Now that you have completed this Unit, check that you are able to use the following in simple programs:

String storage locations ☐
PRINT . . .;. . . ☐
INPUT ". . .";. . . ☐
IF A$ = B$ THEN . . . ☐
Simple sorting procedure ☐

Answers to SAQ's and Exercises

SAQ 1

Valid string locations: A$, K$.
Of the others, M8 and B9 are number store locations.
The rest are neither number nor string store locations.

SAQ 2

(a), (b) and (e) are correct although (e) is not acceptable in minimal BASIC.
(c) is incorrect: M$ is a string location so you need "9583"
(d) is incorrect: K8 is a number store location so the string "JAM POT" cannot be assigned to it.

SAQ 3

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
P	R	I	N	T		L	A	Y	O	U	T										
B	A	S	I	C												C	O	U	R	S	E
B	A	S	I	C	C	O	U	R	S	E											
B	A	S	I	C		C	O	U	R	S	E										

(Answer shown for standard print zones.)

SAQ 4

```
10  LET B$ = "BASIC"
20  PRINT B$, B$
30  PRINT B$; B$; B$; B$
40  PRINT B$; " "; B$; " "; B$
50  STOP
```

Program 10

SAQ 5

```
WHAT IS YOUR NAME? JOHN SMITH
WHAT IS YOUR AGE? 45
THANK YOU JOHN SMITH 45
```

SAQ 6

```
10  INPUT "WHAT IS YOUR NAME"; N$
20  INPUT "WHAT IS YOUR AGE"; A
30  PRINT "MY NAME IS"; N$; " AND I AM "; A; "YEARS OLD"
40  STOP
```

Exercise 1

Exercises 1 and 2 are very similar in nature and an answer for Exercise 1 has not been included.

Exercise 2

```
10  REM★★ENQUIRY LETTER★★
15  PRINT "          ★★★ENQUIRY LETTER★★★"
16  PRINT
17  PRINT
30  PRINT "DETAILS OF ADDRESSEE"
40  PRINT
50  PRINT "NAME . . .? ";
51  INPUT N$
52  PRINT N$
60  PRINT "STREET . . .? ";
61  INPUT S$
62  PRINT S$
70  PRINT "TOWN . . .? ";
71  INPUT T$
72  PRINT T$
80  PRINT
90  PRINT "DATE FOR THIS LETTER? ";
91  INPUT D$
92  PRINT D$
100 PRINT
110 PRINT "DETAILS OF PRODUCT/SERVICE"
120 PRINT
130 PRINT "ITEM OF INTEREST? ";
140 INPUT I$
141 PRINT I$
150 PRINT "SOURCE OF INFORMATION? ";
160 INPUT A$
161 PRINT A$
170 PRINT "DATE OF SOURCE? ";
171 INPUT E$
172 PRINT E$
180 SCROLL
190 SCROLL
200 SCROLL
210 SCROLL
220 PRINT N$
225 SCROLL
230 PRINT S$
235 SCROLL
240 PRINT T$
250 SCROLL
260 SCROLL
270 PRINT D$
280 SCROLL
290 SCROLL
300 PRINT "DEAR SIR,"
310 SCROLL
315 SCROLL
320 PRINT "WILL YOU KINDLY SEND ME DETAILS ";
```

```
330 PRINT "OF ";I$;","
340 PRINT "AS ITEMISED IN THE"
345 SCROLL
350 PRINT A$
355 SCROLL
360 PRINT "DATED ";E$;"."
370 SCROLL
375 SCROLL
380 SCROLL
390 PRINT "YOURS FAITHFULLY,"
400 SCROLL
410 SCROLL
415 SCROLL
420 SCROLL
430 PRINT "O. L. SEYMOUR"
```

Program 11

Typical run

```
RUN
DETAILS OF ADDRESSEE

NAME. . .? E P SOFTWARE LTD
STREET. . .? EDGWARE ROAD
TOWN . . .? LONDON

DATE FOR THIS LETTER? 12TH OCTOBER 1980

DETAILS OF PRODUCT/SERVICE

ITEM OF INTEREST
? BUSINESS SOFTWARE PACKAGES
SOURCE OF INFORMATION
? MAGAZINE "MODERN COMPUTING"
DATE OF SOURCE? 10TH OCTOBER


E P SOFTWARE LTD
EDGWARE ROAD
LONDON

12TH OCTOBER 1980

DEAR SIR,

WILL YOU KINDLY SEND ME DETAILS OF
BUSINESS SOFTWARE PACKAGES,
AS ITEMISED IN THE
MAGAZINE "MODERN COMPUTING"
DATED 10TH OCTOBER.

YOURS FAITHFULLY,


O.L. SEYMOUR
```

SAQ 10

20 PRINT "NUMBER OF PERSON SOUGHT"
35, 45 and 55 change N$ to T$
195 PRINT "SUBSCRIBER NUMBER ";Q$;" IS NOT IN FILE"
200 PRINT "SUBSCRIBER NUMBER ";Q$;" IS ";N$

Exercise 3

A simple method of solving this problem is shown in the Program 12. The names are input one by one, and the 'lowest so far' always stored in A$. The program, however, only retains this one item of information, all other data is lost.

```
10  REM★★FIRST IN ALPHA-ORDER★★
20  PRINT "FIRST NAME? ";
21  INPUT A$
22  PRINT A$
30  PRINT "NEXT NAME? ";
31  INPUT B$
32  PRINT B$
40  IF B$="ZZZZ" THEN GOTO 90
50  IF A$<B$ THEN GOTO 70
60  LET A$=B$
70  PRINT "FIRST IN ALPHA-ORDER SO FAR IS  ";A$
80  GOTO 30
90  PRINT A$;"  WAS OVERALL FIRST"
100 STOP
```

Program 12

```
RUN
FIRST NAME? TOM
NEXT NAME? SID
FIRST IN ALPHA-ORDER SO FAR IS SID
NEXT NAME? JOE
FIRST IN ALPHA-ORDER SO FAR IS JOE
NEXT NAME? PETE
FIRST IN ALPHA-ORDER SO FAR IS JOE
NEXT NAME? FRED
FIRST IN ALPHA-ORDER SO FAR IS FRED
NEXT NAME? BILL
FIRST IN ALPHA-ORDER SO FAR IS BILL
NEXT NAME? RON
FIRST IN ALPHA-ORDER SO FAR IS BILL
NEXT NAME? ALAN
FIRST IN ALPHA-ORDER SO FAR IS ALAN
NEXT NAME? ZZZZ
ALAN WAS OVERALL FIRST
```

Appendix

American Standard Code for Information Interchange or ASCII Code

That part of the code which concerns us here is shown below.

@	100 0000	0	011 0000
A	100 0001	1	011 0001
B	100 0010	2	011 0010
C	100 0011	3	011 0011
D	100 0100	4	011 0100
E	100 0101	5	011 0101
F	100 0110	6	011 0110
G	100 0111	7	011 0111
H	100 1000	8	011 1000
I	100 1001	9	011 1001
J	100 1010	:	011 1010
K	100 1011	;	011 1011
L	100 1100	<	011 1100
M	100 1101	=	011 1101
N	100 1110	>	011 1110
O	100 1111	?	011 1111
P	101 0000	Space	010 0000
Q	101 0001	!	010 0001
R	101 0010	"	010 0010
S	101 0011	#	010 0010
T	101 0100	$	010 0100
U	101 0101	%	010 0100
V	101 0110	&	010 0110
W	101 0111	.	010 0111
X	101 1000	(	010 1000
Y	101 1001	)	010 1001
Z	101 1010	"	010 1010
[	101 1011	+	010 1011
\	101 1100	"	010 1100
]	101 1101	–	010 1101
↑	101 1110	.	010 1110
—	101 1111	/	010 1111

Comment 1

7 electronic circuits each in the on (1) or off (0) state may be used to represent the characters shown.

Comment 2

Without knowing anything about binary representation we can still pick out another useful feature of this code. If we read

A = 'one million and 1'
B = 'one million and 10'
Z = 'one million, eleven thousand and ten'

even though this interpretation is incorrect we can still see that the letters are ranked in order A<B<C . . .<Y<Z. We will find this fact very useful subsequently.

UNIT 4
Lists

4.1	Variables	90
4.2	Lists	90
4.3	List variables	90
4.4	List input and output	92
4.5	The FOR . . . NEXT . . . loop	97
4.6	Nested loops	102
4.7	Interchanging	105
	Assignment 4	109
	Objectives of Unit 4	110
	Answers to SAQ's and Exercises	110

4.1 Variables

We have already seen how a memory location may store several different values during the course of a program's execution. Thus the value in a store location may vary during a run, and so we often refer to the location names as variables in a program. Thus the 286 store labels:

A, A0, A1, . . . A9, B, B0, . . . Z8, Z9

are called numeric variables, and their 26 counterparts:

A$, B$, . . . Z$,

are called string variables. The store labels are used in expressions in program statements just as mathematicians use variables in equations.

4.2 Lists

Lists and supermarkets seem inextricably linked. We go in with a list of items which we wish to buy, and emerge with the items and a list of prices in the form of a receipt. The list of prices results from the process of transferring the items from the basket to the counter. This is a fairly random process but we could have given the list a more meaningful order in a variety of ways. With a lot of effort we could have taken the items out of the basket in order of price, i.e. the cheapest first, the next cheapest next, and so on with the most expensive last, so that the till roll of prices would be in order of cost. Similarly, we could have taken them out of the basket in order of weight, the lightest first through to the heaviest last; and by so doing impose a completely different order on the receipt list. Being able to relate the position in the list in some way to the value of the item, is the most useful feature of lists, as we shall see by the end of this Unit.

4.3 List variables

Most of the data that we have considered so far can be classified into sets. We have considered sets of test marks, sets of names and associated telephone numbers, sets of countries and their capital cities. Most data can be classified in some way. If we are collecting data for some purpose, this very purpose gives the set of values common characteristics. There are obvious advantages to naming the storage locations for items in a set of data in a way which emphasises that all the items belong to one set. Even better, it would be useful if the storage location names identified the position of an item within the set.

For example, storage, or variable, notation which emphasises that the values are in some way associated with each other, and allocates a position within the set. This is achieved in the following way.

Consider a set of marks in a teacher's mark book. They form a natural list and could be allocated storage locations in the following way:

Item	Storage location symbol
The first member of the M-list is 42	M(1) = 42
The second member of the M-list is 67	M(2) = 67
The third member of the M-list is 90	M(3) = 90
etc . . .	

M(1), M(2), M(3) are like separate memory locations. You can take any of the 26 store locations and put numbers in brackets after them to make list store locations, e.g.

List name	List store locations in that list
M(I)	M(1), M(2), M(3) . . .
C$(I)	C$(1), C$(2), C$(3) . . .

The number in the brackets (here shown as I) is called the index of the list, and may be any positive integer within the memory capacity of your computer.

String lists
As you can see from the table above, lists can be string lists as well as numeric lists. Thus if you name a list M(I), it is clearly a list of numbers but M$(I) would be a list of strings, e.g. a list of names could be stored:

Index	Item	Variable name
1	Jones	N$(1)
2	Alan	N$(2)
3	Smith	N$(3)
etc	etc	etc

Figure 1 String list names

Lists and arrays
A table of data, like that shown in Figure 1, is often referred to as an array of data. With the data displayed in rows and columns in this way (indexed by item), a table is often referred to as a two-dimensional array. A list (i.e. just one column of data) is similarly called a one-dimensional array. We will see how BASIC provides for two-dimensional arrays in a later Unit.

DIM or how long is a list?
As long as you choose! We can choose a list to be of any desired length, provided that we warn the system first. We do this with a DIM statement which appears in the program before the array it refers to.

DIM for numerical arrays
If you want to use an array, M say, to store 8 numbers then you announce your intention with the statement:

```
DIM M(8)
```

This tells the computer to reserve memory space for the 8 variables M(1), M(2), M(3), . . . , M(8).

DIM for string arrays

When you want to reserve space for a string array you have to tell the computer two things: (a) the number of strings you wish to store in the array and (b) the maximum length of any one string. You do this by a DIM statement of the form:

```
DIM N$(10, 15)
        |   └maximum string length
        └────number of strings
```

This creates 10 storage locations N$(1), N$(2), N$(3), . . . , N$(10) each capable of holding a string of 15 characters.

Items and index numbers

The following paper and pencil exercises should reinforce your understanding of what is meant by the terms item and index, and their often only fleeting relationship. They also prepare the ground for the interchange-sort procedure which we will consider in detail later in this Unit.

Example 1

Transfer the item of lowest value in the following list to position 1, by comparing in turn each of the values in the remainder of the list with the current value at position 1. Interchange the items if the one in the remainder of the list is lower than that at position 1. (It is easier to do than to describe!)

List: 3, 42, −8, 9, −11

Start		Compare & interchange stages			
position or index	item	1st run c	2nd run c&i	3rd run c	4th run c&i
1	3	3	−8	−8	−11
2	42	42	42	42	42
3	−8	−8	3	3	3
4	9	9	9	9	9
5	−11	−11	−11	−11	−8

Figure 2 Sort to place lowest number first

SAQ 1

Carry out the procedure shown in Example 1 for the following list of numbers:

6,8,4,7,3,9,1.

4.4 List input and output

Before we can manipulate the items in a list we have to get the list of items into the computer, and after processing usually get another list out.

Example 2
Write a BASIC program to input three numbers into a list and output the elements of the list in reverse order.

Solution
We will call the list A(I). The required program is then:

```
10  REM★★EXAMPLE 2★★
15  DIM A (3)
20  INPUTA(1)
30  INPUTA(2)
40  INPUTA(3)
50  PRINT
60  PRINT
70  PRINTA(3),A(2),A(1)
80  STOP
```

Program 1 Reversing the order of a list

Typical run

```
RUN
29
32
−17

−17    32    29
```

🄺 Program 1.

We have done as requested in the question, but have not made a significant advance in programming technique since we could have done the job with the techniques of earlier units simply by calling the variables P, Q and R and outputting them as R, Q and P. To do the job better we need to count the list as it is inputted so that we can use the counter in reverse order when we output the list. The next example adds this refinement.

Counting a list
Example 3
Write a program to input five numbers into a list, to display the items of the list and its index in the form of a table, and then output the elements of the list in reverse order.

Solution
As listed in the question, there are three main parts to the solution and we can display these in flowchart form as in Figure 3.

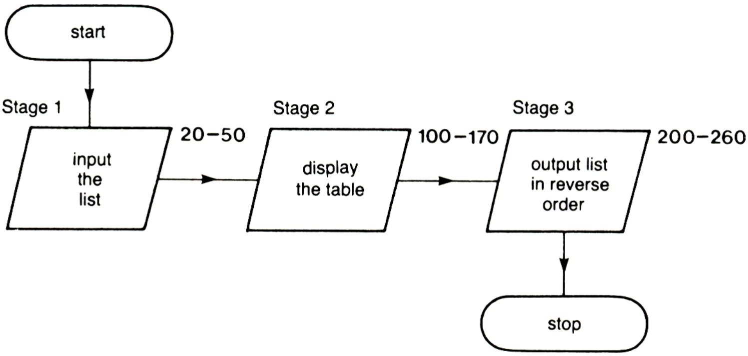


Figure 3 Stages in solving Example 3

Stage 1 We use the variable C to count the elements of the list on input, and to act as an index for the L-list.

```
10  REM★★INPUT A LIST★★
15  DIM L(5)
20  LET C=1
30  PRINT "ENTER THE NEXT NUMBER"
31  INPUT L(C)
32  PRINT L(C)
40  LET C=C+1
50  IF C<=5 THEN GOTO 30
```

Program 2 Counting a list on entry

Stage 2 The table will have the form you met in the answer to SAQ 1. A simple PRINT . . .,. . . will suffice to display the table.

```
100  REM★★DISPLAY THE TABLE★★
110  PRINT
120  PRINT
130  PRINT"INDEX","ITEM"
140  LET C=1
150  PRINT C,L(C)                prints the table
160  LET C=C+1
170  IF C<=5 THEN GOTO 150
```

Program 3 Printing the list in input order

Stage 3 Now we make C count from 5 down to 1 in order to print the list in reverse order.

```
200  REM★★OUTPUT LIST IN REVERSE★★
210  PRINT
220  PRINT
230  LET C=5
240  PRINT L(C)                  prints list in reverse order
250  LET C=C-1
260  IF C>=1 THEN GOTO 240
```

Program 4 Printing in reverse order

We have shown the three program modules that provide the solution. All we have to do now is put them together as follows. The few changes (which don't affect what the program does) are explained in the comments.

	Changes
10 REM★★INPUT A LIST★★	
15 DIM L(5)	
20 LET C=1	
30 PRINT "ENTER THE NEXT NUMBER? ";	
31 INPUT L(C)	
32 PRINT L(C)	
40 LET C=C+1	
50 IF C<=5 THEN GOTO 30	
60 REM 70 REM	REM statements to help reader see the divisions in the program.
100 REM★★DISPLAY THE TABLE★★	
110 PRINT	
120 PRINT	
130 PRINT"INDEX","ITEM"	
140 LET D=1 150 PRINT D,L(D) 160 LET D=D+1 170 IF D<=5 THEN GOTO 150	We've used D as an index to remind you that its name doesn't matter – only its value.
180 REM 190 REM	More REM's
200 REM★★OUTPUT LIST IN REVERSE★★	
210 PRINT	
220 PRINT	
230 LET E=5 240 PRINT L(E) 250 LET E=E−1 260 IF E>=1 THEN GOTO 240	And we've used E here.
270 STOP	END added.

Program 5 The full reverse list program

```
RUN
ENTER THE NEXT NUMBER? -8
ENTER THE NEXT NUMBER? 15
ENTER THE NEXT NUMBER? 23
ENTER THE NEXT NUMBER? -4
ENTER THE NEXT NUMBER? 19

INDEX          ITEM
1              -8
2              15
3              23
4              -4
5              19

19
-4
```

23
15
−8

[K] Program 5.

To solve the problem in Example 3 we've done quite a bit of programming but the program only works for a list of five numbers. A small reward for a great effort! But if we change statements 20–50 which counted the five items, we can make the program accept any number of items so long as we know the number before inputting.

But we need to make DIM M$ depend on the number of items we are inputting. Because the counter C will go up to N+1 (watch line 50), DIM M$ needs to be N+1.

```
10  REM★★INPUT A LIST OF N ITEMS★★
20  PRINT "HOW MANY ITEMS IN THE LIST ";
21  INPUT N
22  PRINT N
25  DIM M (N+1)
30  LET C=1
40  PRINT "ENTER THE NEXT ITEM ";
41  INPUT M(C)
42  PRINT M(C)
50  LET C=C+1
60  IF C<=N THEN GOTO 40
```

any number of items can now be entered into the list M (C)

Program 6

To be able to input any number of items of data into individual store locations with just half a dozen statements represents a significant improvement in programming technique. But, of course, we are never satisfied! Why should we bother to count the items, especially if the list is long, when we can get the computer to do it for us? The next exercise asks you to do this.

Exercise 1
Write a BASIC program to (a) input a list of numbers of unknown length; terminate the list with the dummy '−9999'. Call the list P(C) and assume that the list will be 30 or less items. So DIM P (31) will declare the list. Check your answer.
(b) Now modify your program to output those items whose index is odd.

4.5 The FOR . . . NEXT . . . loop

As we have said (repeatedly!) a computer is good at lots of repetitive operations. In order to control these operations we usually have to count them. Since we introduced the idea of counting in Unit 2, we have used the following sequence of statements several times.

```
LET C=1

activity

LET C=C+1
IF C<=N THEN return to activity
```

Program 7

These repetitive operations are so important in programming that special provision is made for them. In BASIC this is done by means of the FOR . . . NEXT . . . facility.

The sequence in Program 7 has three elements:

LET C = 1	which starts the count
LET C=C+1	which defines the incremental step (1 in this case; 2 in line 270 of the answer to Exercise 1)
C<=N	which stops the counting process

The same features occur in the FOR . . . NEXT . . . loop where the above sequence becomes:

```
              start   finish   increment
             /        /        /
FOR C=1 TO N STEP (1)

activity

NEXT C
```

Program 8

Notice that NEXT C returns control to the line FOR C=1 TO N STEP (1) without you having to put the line number of FOR C . . . in the NEXT C statement.

Arrays and FOR. . . NEXT loops were made for each other. Together they form possibly the most potent facility of the BASIC language. Let's look in detail at how these loops work, and then repeat some of our earlier routines with lists using this new facility.

Examples of FOR. . . NEXT loops in action

(a)

```
10   FOR I=4 TO 10 STEP (2)
20   PRINT I
30   NEXT I
40   STOP
RUN
4
6
8
10
```

(c)

```
10   FOR J=−3 TO 10 STEP(3)
20   PRINT J
30   NEXT J
40   STOP
RUN
−3
0
3
6
9
```

(b)

```
10   FOR K=11 TO 4 STEP(−2)
20   PRINT K
30   NEXT K
40   STOP
RUN
11
9
7
5
```

(d)

```
10   FOR L=4 TO −5 STEP(−2)
20   PRINT L
30   NEXT L
40   STOP
RUN
4
2
0
−2
−4
```

Programs 9–12

SAQ 2

Write out the lists of numbers printed out by the following loops.

(a)

```
10   FOR E=1 TO 9 STEP(2)
20   PRINT E
30   NEXT E
40   STOP
```

(c)

```
10   FOR G=8 TO −4 STEP(−5)
20   PRINT G
30   NEXT G
40   STOP
```

(b)

```
10   FOR F=−30 TO −18 STEP(3)
20   PRINT F
30   NEXT F
40   STOP
```

(d)

```
10   FOR H=−2 TO −11 STEP(−4)
20   PRINT H
30   NEXT H
40   STOP
```

Programs 13–16

FOR. . . NEXT. . . with STEP (1)

The above examples and SAQ's had steps of 2, 3, −2, −4, and −5. Quite often, however, we simply want to use a step of 1. When that is the case, you can omit STEP(1) from the statement. Thus

```
FOR C=1 TO N
   activity
NEXT C
```

Program 17

is taken by the computer to mean a step of 1.

Input/output routines with FOR. . . NEXT
Routines are made easier by the FOR. . . NEXT facility. For example we can rewrite Program 5 using these loops. Notice that at lines 20 and 150 we require a step of 1 so STEP(1) has been omitted.

Comparison with Program 5

```
10   REM★★INPUT A LIST OF 5 NAMES★★
15   DIM L$(5,10)
20   FOR C=1 TO 5
30   PRINT "ENTER THE NEXT NAME? ";  ┐
31   INPUT L$(C)                      ├──── Replaces lines 20–50
32   PRINT L$(C)                      ┘
40   NEXT C
50   REM
60   REM
100  REM★★DISPLAY THE TABLE★★
110  PRINT
120  PRINT
130  PRINT"INDEX","NAME"
140  PRINT
150  FOR D=1 TO 5      ┐
160  PRINT D, L$(D)    ├──── Replaces lines 140–170
170  NEXT D            ┘
180  REM
190  REM
200  REM★★OUTPUT LIST IN REVERSE ORDER★★
210  PRINT
220  PRINT
230  FOR E=5 TO 1 STEP(−1)  ┐
240  PRINT E,L$(E)          ├──── Replaces lines 230–260
250  NEXT E                 ┘
280  STOP
```

Program 18 Using FOR. . . NEXT. . . to reverse a list

```
RUN
ENTER THE NEXT NAME? DICKENS
ENTER THE NEXT NAME? HARDY
ENTER THE NEXT NAME? SNOW
ENTER THE NEXT NAME? AUSTEN
ENTER THE NEXT NAME? ORWELL
```

INDEX	NAME
1	DICKENS
2	HARDY
3	SNOW
4	AUSTEN
5	ORWELL
5	ORWELL
4	AUSTEN
3	SNOW
2	HARDY
1	DICKENS

[K] Program 18.

General FOR. . . NEXT loop
The FOR. . . NEXT loop can be expressed in quite general terms as

```
FOR I = S TO F STEP (J)
    activity
NEXT I
```

providing that S, F and J are given 'reasonable' values before the loop is executed. Unreasonable values would be something like

S=2, F=10, and STEP(−3)

since you can't get from 2 to 10 in steps of −3, in the normal course of events.

Exercise 2
In Exercise 2 of Unit 2 you wrote a program (Program 19) to calculate the yield on an investment for a period of years to be specified. Re-write lines 40–90 of the program using a FOR. . . NEXT loop.

Exercise 3
If you are mathematically inquisitive you might like to try the following which demonstrates the potential of the FOR. . . NEXT loop. Write a program to tabulate the squares and cubes of the odd integers from 1 to 21 inclusive.

Output display
We always aim at a clear presentation of output data on the screen or printer. The FOR. . . NEXT facility is used widely in presenting output routines.

Use to skip lines. As you saw in the layout of letters in Unit 3 it is useful to 'print' blank lines. The following routine does this for you.

```
10  REM★★FOR. . . .NEXT★★
20  REM★★TO SKIP LINES IN A PRINT ROUTINE★★
30  PRINT "HELLO"
40  FOR H=1 TO 10      instruction to print 10 blank lines
```

```
50  PRINT
60  NEXT H
70  PRINT"HELLO FROM 11 LINES BELOW"
80  STOP
```

Program 19

```
RUN
HELLO









HELLO FROM 11 LINES BELOW
```

K Program 19.

Drawing a line. We may wish to print lines across the screen or page, e.g. to separate blocks of data or just to underline. The following routine does this.

```
10  REM★★FOR. . . .NEXT★★
20  REM★★TO DRAW LINES★★
30  FOR M=1 TO 40
40  PRINT"★";              instruction to print ★ 40 times
50  NEXT M
60  PRINT
90  STOP
```

Program 20

```
RUN

★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
```

SAQ 3
Why does line 40 in Program 20 have ; at the end of the line? What happens if line 40 is 40 PRINT "★"?

K Program 20.

Loops in flowcharts
Loops are so important that they have a special flowchart symbol of their own:

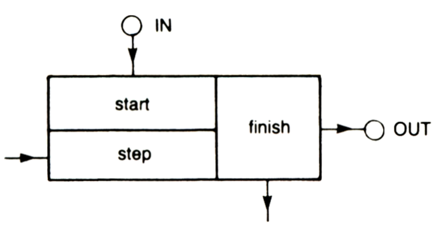


Figure 4a Flowchart symbol for a FOR. . . NEXT. . . loop

The floating ends are the connections to the activity:

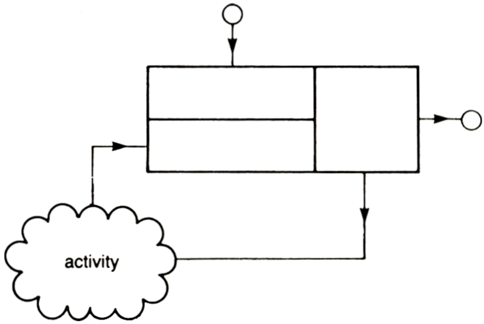


Figure 4b Flowchart symbol's relationship to activity

4.6 Nested loops

Program 20 drew a line of 40 asterisks. We can re-write the program so that we can specify in line 30 a number of asterisks and so vary the length of the line. So with the program:

```
10  REM★★LINES OF DIFFERENT LENGTH★★
20  LET N=1                        number to go in line 30
30  FOR M=1 TO N
40  PRINT "★";
50  NEXT M
60  PRINT
70  STOP
```

Program 21

we get

```
RUN
★
```

To vary the line length we simply key in

20 LET N= required length

and key run:

```
20 LET N=2

RUN
★★

20 LET N=3

RUN
★★★
```

```
20 LET N=4

RUN
★★★★

20 LET N=8

RUN
★★★★★★★★

20 LET N=16

RUN
★★★★★★★★★★★★★★★★

20 LET N=32

RUN
★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
```

[K] Program 21.

Nested FOR. . . NEXT loops
You have seen in Program 21 how you can control the effect of the FOR. . . NEXT loop of lines 30–50 by changing the value of N. We hope by now that the 'obvious' question springs to your mind: 'Why not control the value of N by using another FOR. . . NEXT loop?' The following program does just that. The M-loop of lines 30–50 is itself controlled by the N-loop of lines 20–70. The M-loop is said to be 'nested' within the N-loop.

```
10  REM★★NESTED FOR. . . NEXT LOOPS★★
20  FOR N=1 TO 16
30  FOR M=1 TO N
40  PRINT"★";
50  NEXT M
60  PRINT
70  NEXT N
80  STOP
```

Inner loop: controls number of ★ in each row

Outer loop: controls the rows

Program 22 Nested loops

```
RUN
★
★★
★★★
★★★★
★★★★★
★★★★★★
★★★★★★★
★★★★★★★★
★★★★★★★★★
```

```
***********
************
*************
**************
***************
****************
*****************
READY
```

It is important that you understand how this works. For example, when the computer has just finished printing the 9th row of asterisks N will be 9. It leaves the inner loop (line 50) and control goes to the outer loop (line 60). A blank line is printed and N then increases to its next value of 10. Control reverts to the inner loop at line 30. The computer then goes around the inner loop (lines 30–50) ten times before exiting again to line 60.

[K] Program 22.

More print patterns from loops
If generating print patterns with loops appeals to you, here is another one plus two Exercises.

```
10  REM**NESTED FOR. . . NEXT LOOPS**
15  LET P=1
20  FOR N=1 TO 5
25  LET P=2*P        Multiplies current value of
30  FOR M=1 TO P     P by 2 for each pass
40  PRINT"*";        around loop.
50  NEXT M
60  PRINT
70  NEXT N
80  STOP
```

Program 23

```
RUN
**                                        P= 2
****                                      P= 4
********                                  P= 8
****************                          P=16
********************************          P=32
```

[K] Program 23.

Exercise 4
Write a program using nested loops to print out the 7, 8 and 9 multiplication tables.

Exercise 5
Write a program using nested loops to print out rectangles of asterisks of dimensions to be chosen by the user.

4.7 Interchanging

We considered the problem of finding the smaller of 2 numbers in Unit 2, and the smallest of 3 in Unit 3. In this Unit we have done exercises on interchanging items of a list, in preparation for writing an interchange program.

We have been comparing the items of a list with that at position 1, and interchanging if the item in the list is smaller than that at position 1. When you did this in SAQ 1, you did it manually and we have not yet looked at the problems of writing a program to perform the interchange. We can't just say

Copy A into B and then B into A

since the first transaction 'copy A into B' overwrites and destroys what's in B, giving us copies of A in A and in B. Instead we have to put the contents of B away in some safe temporary store before we overwrite B with A. We can show the process diagrammatically:

TEMP←B
B←A
A←TEMP

Where ← means place the number in the right hand store into the left hand store.

Suppose, for example, you want to sort a list of names N$-list and you want to interchange the first name, N$(1) with some other name N$(K) at position K. This can be done using a temporary store location T$:

T$←N$(K)
N$(K)←N$(1)
N$(1)←T$

Remember that it is the contents of N$(1) and N$(K) that are being swapped. Suppose N$(1)=FRED and N$(K)=JIM then this is what is happening:

	Store locations		
	N$(1)	N$(K)	T$
start	FRED	JIM	
next stage	FRED	JIM	JIM
next stage	FRED	FRED	JIM
end	JIM	FRED	JIM

(The fact that T$ still has JIM in it doesn't matter: we have achieved the object which is to swap the locations of FRED and JIM.)

Flowchart for name sort

In the number sort (SAQ 1) we wanted to put the lowest number at the top of the list. So we test each name in turn against the one currently at the top of the list and

interchange only if the name under test comes before the one at the top of the list. A flowchart for this is:

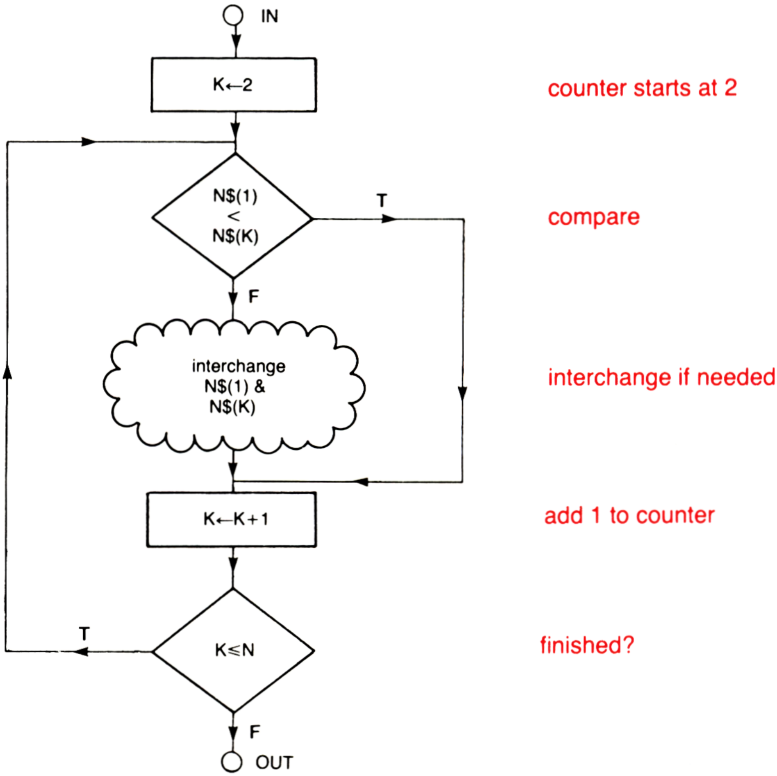


Figure 5a Flowchart for interchange

Or, if we want to use the special flowchart symbol for FOR. . . NEXT. . . , it would look like:

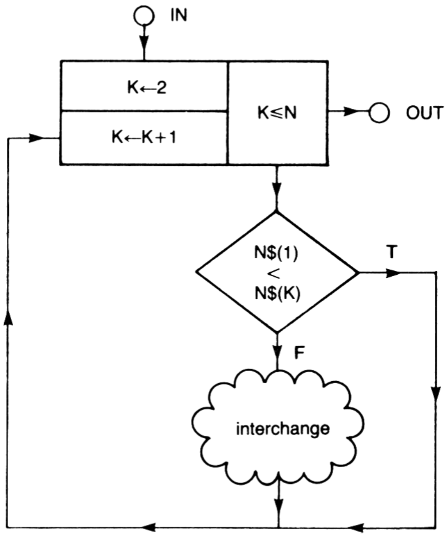


Figure 5b Flowchart for interchange with FOR. . . NEXT. . . symbol

This new routine can now be used to construct a program.

Example 4

Write a program to enter a list of names of unknown length into an array, print out this list with index in input order. By means of the interchange routine place the name of lowest alphabetic value in position 1 in the list, and output the new list.

Solution

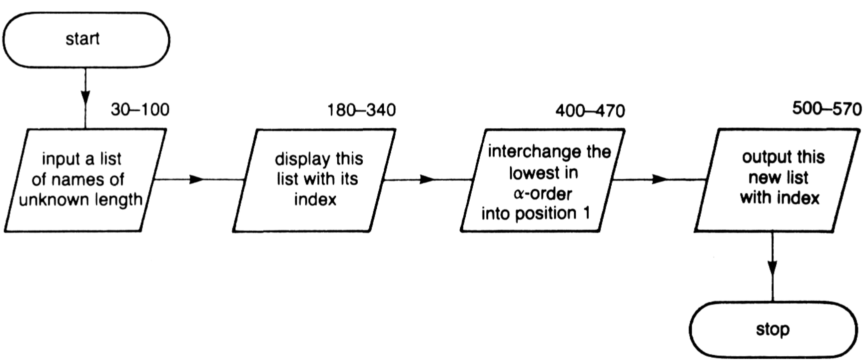


Figure 6 Flowchart for Example 4

Interchange program

```
10   REM★★FIRST IN ALPHA-ORDER★★
20   CLEAR 100
25   DIM N$(20,6)
30   PRINT"ENTER A LIST OF NAMES ONE BY ONE"
40   PRINT"END THE LIST WITH ZZZZ"
50   PRINT
60   LET I=1
70   PRINT "NEXT NAME"
71   INPUT N$(I)                                inputting list
72   PRINT N$(I)
80   IF N$(I) = "ZZZZ" THEN GOTO 200
90   LET I=I+1
100  GOTO 70
180  REM★★★★★★★★★★★★★★
190  REM★★WE DON'T WANT ZZZZ IN OUR LIST, SO★★   finding number
200  LET N=I−1                                    of names
210  REM★★★★★★★★★★★★★★                            entered
300  PRINT
310  PRINT"INDEX","ITEM"
320  FOR J=1 TO N
330  PRINT J, N$(J)                             print list
340  NEXT J
400  REM★★★★★★★★★★★★★
410  REM★★INTERCHANGE ROUTINE★★
420  FOR K=2 TO N
430  IF N$(1)<N$(K) THEN GOTO 470
440  LET T$=N$(K)                               interchange
450  LET N$(K)=N$(1)
460  LET N$(1)=T$
470  NEXT K
500  REM★★★★★★★★★★★★★
510  PRINT
520  PRINT"LIST AFTER INTERCHANGE"
530  PRINT
540  PRINT"INDEX","ITEM"
550  FOR L=1 TO N
560  PRINT L,N$(L)                              print interchanged order
570  NEXT L
580  STOP
```

Program 24 Finding the first item in alphabetical order

Interchange program runs

```
RUN
ENTER A LIST OF NAMES ONE BY ONE
END THE LIST WITH ZZZZ

NEXT NAME? JONES
NEXT NAME? PRICE
```

```
NEXT NAME? DAVIES
NEXT NAME? EVANS
NEXT NAME? ZZZZ

INDEX         ITEM
1             JONES
2             PRICE
3             DAVIES
4             EVANS

LIST AFTER INTERCHANGE

INDEX         ITEM
1             DAVIES
2             PRICE
3             JONES
4             EVANS
```

If a print routine is inserted into the interchange procedure as below (lines 470–478), then we can look at the effect of each pass round the loop.

```
400  REM★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
410  REM★★INTERCHANGE ROUTINE★★
420  FOR K=2 TO N
430  IF N$(1)<N$(K) THEN 470
440  LET T$=N$(K)
450  LET N$(K)=N$(1)
460  LET N$(1)=T$
470  PRINT
472  FOR L=1 TO N           ┐
474  PRINT N$(L);"  ";      │
476  NEXT L                 ├── print routine to observe pass round the loop
478  PRINT                  │
480  NEXT K                 ┘
500  REM★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
```

[K] Program 24 with lines 400 to 500 as above.

Assignment 4

1. It will probably have occurred to you by now that, having placed the item of lowest value into position 1, we could repeat the procedure by placing the item of lowest value in the remainder of the list into positon 2, and so on for the rest of the list. The sort of the complete list in this way demands nested FOR. . . NEXT. . . loops.

Modify Program 24 to sort a complete list into alphabetical order.

2. Input a file of names and associated telephone numbers into two lists N$(I) and T$(I) respectively. Use the index I to search through the file to find a particular name, and if found then to output the associated telephone number.

Objectives of Unit 4

Now that you have completed this Unit, check that you are able to write simple programs using:

List store location names
- to input lists ☐
- to print lists ☐

Counters to count the number of items in a list. ☐

FOR. . . NEXT. . . loops
- to print a list ☐
- to input a list ☐
- to print ★ layouts ☐

Nested loops
- to print ★ layouts ☐

Interchange routine ☐

Answers to SAQ's and Exercises

SAQ 1
The six stages of the procedure are shown here in the following program run:

```
RUN
ENTER A LIST OF NAMES ONE BY ONE
END THE LIST WITH ZZZZ


NEXT NAME? 6
NEXT NAME? 8
NEXT NAME? 4
NEXT NAME? 7
NEXT NAME? 3
NEXT NAME? 9
NEXT NAME? 1
NEXT NAME? ZZZZ


INDEX    ITEM
1        6
2        8
3        4
4        7
5        3
6        9
7        1
```

```
6  8  4  7  3  9  1
4  8  6  7  3  9  1
4  8  6  7  3  9  1
3  8  6  7  4  9  1
3  8  6  7  4  9  1
1  8  6  7  4  9  3
```

LIST AFTER INTERCHANGE

INDEX	ITEM
1	1
2	8
3	6
4	7
5	4
6	9
7	3

Exercise 1
Notice that we've used lots of REM statements to tell you how the program works.

```
5   DIM P(31)
10  REM★★A LIST OF NUMBERS OF UNKNOWN LENGTH★★
20  PRINT"ENTER THE ELEMENTS OF THE LIST"
22  PRINT"ITEM BY ITEM AS REQUESTED"
24  PRINT"END THE LIST WITH THE DUMMY '−9999'"
26  PRINT
30  LET C=1
40  PRINT "ENTER THE NEXT NUMBER? ";
41  INPUT P(C)
42  PRINT P(C)
50  IF P(C)=−9999 THEN GOTO 100
```

Input sequence

```
40  INPUT"ENTER THE NEXT NUMBER";P(C)
50  IF P(C)=−9999 THEN 100
60  LET C=C+1
70  GOTO 40
80  REM★★★★★★★★★★★★★★
90  REM★★REMEMBER 'C' COUNTED −9999 AS AN ITEM★
100 LET N=C−1
```

Taking correct total from counter

```
110 REM★★★★★★★★★★★★★★
120 REM
130 REM
200 REM★★OUTPUT THE ITEMS WHOSE INDEX IS ODD★★
210 REM★★3=1+2..5=3+2..7=5+2..ETC. . . . . . . . . . . .★★
220 LET C=1
230 PRINT
240 PRINT
250 PRINT"ODD INDEX","ITEM"
260 PRINT C,P(C)
```

Output sequence

```
270  LET C=C+2
280  IF C<=N THEN GOTO 260
290  STOP
```

Program 25

```
RUN
ENTER THE ELEMENTS OF THE LIST
ITEM BY ITEM AS REQUESTED
END THE LIST WITH THE DUMMY '-9999'

ENTER THE NEXT NUMBER? 42
ENTER THE NEXT NUMBER? -12
ENTER THE NEXT NUMBER? 37
ENTER THE NEXT NUMBER? 92
ENTER THE NEXT NUMBER? 11
ENTER THE NEXT NUMBER? -3
ENTER THE NEXT NUMBER? -9999

ODD INDEX       ITEM
1               42
3               37
5               11
```

SAQ 2

(a) 1, 3, 5, 7, 9.
(b) −30, −27, −24, −21, −18.
(c) 8, 3, −2.
(d) −2, −6, −10.

Exercise 2

```
10   REM★★COMPOUND INTEREST★★
20   PRINT "ENTER YEARS ";
30   INPUT N
40   PRINT N
50   PRINT "ENTER DEPOSIT ";
60   INPUT D
70   PRINT D
80   PRINT "ENTER INTEREST ";
90   INPUT I
100  PRINT I
120  FOR C=1 TO N
130  LET Y=(P★D)/100
140    PRINT "YEAR ";C;"YIELD ";Y          The FOR . . . NEXT . . . loop
150    LET D=D+Y
160    NEXT C
170  STOP
```

Program 26

```
RUN
ENTER YEARS 5
ENTER DEPOSIT 500
ENTER INTEREST 11.25
```

```
YEAR  1          YIELD  56.25
YEAR  2          YIELD  62.5781
YEAR  3          YIELD  69.6182
YEAR  4          YIELD  77.4502
YEAR  5          YIELD  86.1634
```

Exercise 3

```
10  REM★★SQUARES AND CUBES★★
20  PRINT "NUMBER - - - - SQUARE - - - - CUBE"
30  FOR I=1 TO 21 STEP(2)
40  LET S=I★I
50  LET C=I★I★I
60  PRINT I;
62  PRINT " - - - - - - - - - - ";S;
64  PRINT " - - - - - - - - - - - ";C
70  NEXT I
80  STOP
```

Program 27

```
RUN
NUMBER          SQUARE          CUBE
1               1               1
3               9               27
5               25                125
7               49                343
9               81                729
11                121               1331
13                169               2197
15                225               3375
17                289               4913
19                361               6859
21                441               9261
```

The display is a bit of a mess, but we will fix that in the next Unit with TAB.

SAQ 3

; suppresses the print return so that the print head stops after printing ★. Thus the next ★ will be printed on the same line. Without ; the asterisks would be printed in a column 40 print lines deep.

Exercise 4

```
10  REM★★MULTIPLICATION TABLES★★
40  FOR T=7 TO 9
50  FOR K=1 TO 12
60  LET P=K★T
70  PRINT K;"TIMES";T;"=";P
80  NEXT K
90  PRINT
100 NEXT T
110 STOP
```

Program 28

```
RUN
1 TIMES 7 = 7
2 TIMES 7 = 14
3 TIMES 7 = 21
4 TIMES 7 = 28
5 TIMES 7 = 35
6 TIMES 7 = 42
7 TIMES 7 = 49
8 TIMES 7 = 56
9 TIMES 7 = 63
10 TIMES 7 = 70
11 TIMES 7 = 77
12 TIMES 7 = 84

1 TIMES 8 = 8
2 TIMES 8 = 16
3 TIMES 8 = 24
4 TIMES 8 = 32
5 TIMES 8 = 40
6 TIMES 8 = 48
7 TIMES 8 = 56
8 TIMES 8 = 64
9 TIMES 8 = 72
10 TIMES 8 = 80
11 TIMES 8 = 88
12 TIMES 8 = 96

1 TIMES 9 = 9
2 TIMES 9 = 18
3 TIMES 9 = 27
4 TIMES 9 = 36
5 TIMES 9 = 45
6 TIMES 9 = 54
7 TIMES 9 = 63
8 TIMES 9 = 72
9 TIMES 9 = 81
10 TIMES 9 = 90
11 TIMES 9 = 99
12 TIMES 9 = 108
```

we still have a lot to learn about 'tabulation'.

Exercise 5

```
10  PRINT "LENGTH OF RECTANGLE "
11  INPUT L
12  PRINT L
20  PRINT "WIDTH OF RECTANGLE ";
21  INPUT W
22  PRINT W
```

```
30  FOR I = 1 TO W
40  FOR J = 1 TO L
50  PRINT "★";
60  NEXT J
70  PRINT
80  NEXT I
90  STOP
```

UNIT 5

An end to strings and PRINT

5.1	Introduction	118
5.2	Length of a string of characters	118
5.3	Frequency tables	119
5.4	Frequency diagrams	123
5.5	Tabulation	125
5.6	Cutting up strings	129
5.7	VAL	133
	Assignment 5	137
	Objectives of Unit 5	137
	Answers to Exercises and SAQ's	137

5.1 Introduction

The earlier units were concerned with introducing topics; new ideas came thick and fast. This Unit is mainly concerned with strings, but you will meet the TAB statement which is an important addition to your printing repertoire. The title of this Unit is a slight exaggeration, but by the end of this Unit you will have met most of the main string and print functions of the BASIC language.

5.2 Length of a string of characters

We asked the question 'How long is a piece of string?' in Unit 3. At the time it may have seemed a rather facetious question, but the number of characters contained in a particular string storage location is often a vital piece of information. This is especially so if we are trying to use the memory allocation of a particular computer as efficiently as possible.

In BASIC the operation LEN(A$) gives the length of A$ as a number of characters. Thus:

If A$="FRED" then LEN (A$) = 4
If B$="I" LEN (B$) = 1

SAQ 1
What are the values of the following:

(a) LEN (C$) where C$= "ANN"
(b) LEN (D$) where D$ = "A"
(c) LEN (E$) where E$ = "72"
(d) LEN (F$) where F$ = "CAT 123"

Example 1
Write a BASIC program to input a list of ten words, ending with ZZZZ, and to print out the length of each word.

Solution
We have set up an input routine (lines 98 to 102) to input the words into a string list, W$(I). For convenience we have limited the maximum word length to 10 so DIM W$ is DIM W$(10,10).

```
10  REM★★LENGTH OF A WORD★★
20  REM★★★★★★★★★★★★★
30  REM★★INPUT WORDS ONE BY ONE★★
40  REM★★AND OUTPUT THEIR LENGTHS★★
90  REM★★★★★★★★★★★★★
97  DIM W$(10,10)
98  FOR J=1 TO 10
99  PRINT "NEXT WORD";
100 INPUT W$(I)
101 PRINT W$(I)
102 NEXT I
110 IF W$="ZZZZ" THEN 200
120 LET L=LEN(W$)
130 PRINT
```

READ, LEN, PRINT cycle

```
140 PRINT W$;" HAS ";L;" LETTERS"
150 GOTO 100
200 STOP
```

Program 1 Measuring word lengths

```
RUN
DEVISE HAS 6 LETTERS

AN HAS 2 LETTERS

ALGORITHM HAS 9 LETTERS

AND HAS 3 LETTERS

WRITE HAS 5 LETTERS

A HAS 1 LETTERS

BASIC HAS 5 LETTERS

PROGRAM HAS 7 LETTERS
```

(Result when DEVISE, AN, ALGORITHM, AND, WRITE, A, BASIC, PROGRAM, ZZZZ was inputted).

[K] Program 1.

5.3 Frequency tables

Measuring the frequency with which something occurs is commonly needed in handling numerical information. For example, a knowledge of the frequency with which certain letters occur in normal language usage is an important factor in code-breaking activities. In order to measure frequencies it is useful to be able to use the simple technique used in statistical analysis of tally marks. This first paper and pencil example introduces this.

Tally marks

Example 2

Find the frequency with which each vowel occurs in the following words.

THE, HORSE, STOOD, STILL, TILL, HE, HAD, FINISHED, THE, HYMN, WHICH, JUDE, REPEATED, UNDER, THE, SWAY, OF, A, POLYTHEISTIC, FANCY, THAT, HE, WOULD, NEVER, HAVE, THOUGHT, OF, HUMOURING, IN, BROAD, DAYLIGHT

Solution

There are two ways to approach the problem.

(a) Go through crossing out and counting up all the A's, and then through again counting the number of E's, etc. This would involve 5 passes through the data for fairly sparse information (i.e. for a low hit-rate);

(b) Draw up a table as below and take each vowel in sequence:

TH~~E~~: put a tally mark in the E row;

H~~O~~RS~~E~~: put a mark in the O row, followed by another in the E row;

ST~~O~~~~O~~D: put two more marks in the O row.

Vowel	Count				Total count or frequency
A	~~1111~~	1111			9
E	~~1111~~	~~1111~~	~~1111~~	1	16
I	~~1111~~	~~1111~~			10
O	~~1111~~	~~1111~~			10
U	~~1111~~	1			6

TH~~E~~, H~~O~~RS~~E~~, ST~~O~~~~O~~D, ST~~I~~LL, T~~I~~LL, H~~E~~, H~~A~~D, F~~I~~N~~I~~SH~~E~~D, TH~~E~~, HYMN, WH~~I~~CH, J~~U~~D~~E~~, R~~E~~P~~E~~~~A~~T~~E~~D, ~~U~~ND~~E~~R, TH~~E~~, SW~~A~~Y, ~~O~~F, ~~A~~, P~~O~~LYTH~~E~~~~I~~ST~~I~~C, F~~A~~NCY, TH~~A~~T, H~~E~~, W~~O~~~~U~~LD, N~~E~~V~~E~~R, H~~A~~V~~E~~, TH~~O~~~~U~~GHT, ~~O~~F, H~~U~~M~~O~~~~U~~R~~I~~NG, ~~I~~N, BR~~O~~~~A~~D, D~~A~~YL~~I~~GHT

Figure 1 Completed tally count

SAQ 2
Use the tally method to draw up a frequency table of the lengths of words for the data in Example 2.

Getting the computer to count
Having found a paper and pencil method of counting frequencies, we now need a method of getting the computer to do the counting of a list. The power of lists is derived from an apt use of the index. In question 2 of Assignment 4 we saw how the items of two lists of data (name and telephone number lists) were linked by a common index. The 3rd member of the number-list was the telephone number for the 3rd name in the name-list, etc. Generally the I-th member of the name-list is linked to the I-th member of number-list.

Suppose we want to count the number of times the digits 0, 1, 2, . . . 9 occur in a sequence. We can use 10 counters:

C(0), C(1), C(2) . . .   C(9)

each of which will be zero at the start. To count the digits in 473808 we take the first digit in the sequence: 4. 1 is added to C(4) and so on:

Digits entered	Counters after entry									
	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	C(6)	C(7)	C(8)	C(9)
start	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0	1	0	0
3	0	0	0	1	1	0	0	1	0	0
8	0	0	0	1	1	0	0	1	1	0
0	1	0	0	1	1	0	0	1	1	0
8	1	0	0	1	1	0	0	1	2	0

So the idea is that when I is entered at the key-board, increment C(I) by 1. This can usually be achieved in just two BASIC statements.

```
120 INPUT I
140 LET C(I)=C(I)+1
```

A list counter

However the ZX81 does not allow us to use C(0) as a location – locations must start at 1. e.g. C(1), C(2), C(3), . . . etc. We can get around this problem by using

C(1) to count 0s
C(2) to count 1s
C(3) to count 2s, etc.

Thus C(N) counts the N+1s so when we input N, we increment counter C(N+1).

SAQ 3
The sequence

```
10  INPUT N
20  LET C(N+1)=C(N+1)+1
```

is used to count the number of 0's, 1's, 2's, etc. in the following input data: 3, 1, 0, 5, 9, 9, 6, 6, 6, 0, 4, 4, 2, 4, 1, 2, 1, 3, 0, 2, 1, 3. What are the values of the following:

(a) C(4) after 3 numbers have been inputted.
(b) C(10) after 12 numbers have been inputted.
(c) C(2) after all the numbers have been inputted.
(d) C(1) after all the numbers have been inputted.

We will now use this method of counting a list in an example.

Example 3
Write a program to input a sequence of single digits and to output the frequency with which each digit occurs.

Solution
A digit is one of the set of 10 numbers 0, 1, 2, 3 . . . 9. We will enter these one by one, with the sequence being terminated by −9999. So far, very routine!
The counting list will have 10 counters:

C(1), C(2), C(3) . . .   C(10)

The program to solve the complete problem has two parts. (i) The input and increment routine incorporates the two statements 120 and 140 discussed above. (ii) The output routine is driven by a FOR. . . NEXT loop, with index J running from 0 to 9.

```
10  REM★★COUNT THE NUMBER OF TIMES EACH DIGIT IS ENTERED★★
20  REM★★AND STORE IN A COUNT-LIST C(I)★★
30  REM★★★★★★★★★★★★★★★★★
40  DIM C(10)
100 PRINT "INPUT A LIST OF SINGLE DIGITS"
110 PRINT "END THE LIST WITH −9999"
115 PRINT
120 PRINT "NEXT DIGIT"
121 INPUT I                         input and counting routine
122 PRINT I
130 IF I=−9999 THEN 200
```

```
140 LET C(I+1)=C(I+1)+1
150 GOTO 120
190 REM★★★★
200 CLS
205 PRINT
210 PRINT"DIGIT","COUNT"
220 PRINT
230 FOR J=1 TO 10
240 PRINT J-1,C(J)
250 NEXT J
260 STOP
```

printing table

Program 2 Counting with a list counter C(I)

Typical output

(After entering 3, 7, 6, 4, 9, 1, 4, 9, 2, 7, 8, 0, 1, 5, 2, 7, −9999.)

DIGIT	COUNT
0	1
1	2
2	2
3	1
4	2
5	1
6	1
7	3
8	1
9	2

K Program 2.

Frequency table for string lengths

We have written two programs so far in this Unit: the first to find the lengths of strings, and the second to build up a frequency table.

In the following exercise we want you to combine these two ideas to build up a frequency table of lengths of words. If you wish you can use the words in the DATA statements already used in Example 2. Assume that the words will not be longer than 15 characters, so the length-list will have elements:

L(1), L(2), L(3) . . . L(15).

Exercise 1

Write a program to read in a set of words and to display a frequency table of their lengths.

5.4 Frequency diagrams

Frequency diagram for number of vowels

The picture of tally marks in Figure 1 makes a more immediate impact on us and somehow gives us more information about the distribution of frequencies of the vowels than just the column of figures. So why not get the computer to print a picture for us? You saw how to print rows of asterisks in Unit 4 by driving the print head across the page (or screen) with a FOR. . . NEXT loop of variable range.

SAQ 4

What will appear on the screen as a result of the following program if you input 2, 5, 7, 8, 3 and 1?

```
10  INPUT A
20  FOR I=1 TO A
30  PRINT "★";
40  NEXT I
45  PRINT
50  GOTO 10
```

Program 3

We can do the same thing using the frequencies from Figure 1 to determine the range and thus the number of asterisks printed across the page. This will generate a picture of the distribution.

Example 4

Write a program to print out a frequency diagram for the distribution of vowels given in Example 2.

Solution

Notice that this program draws the diagram from the frequencies we have already calculated. We input these frequencies in the lines 40 to 80.

We read the frequencies (lines 50 to 80) with a counter F(K) where F(1) is the number of a's, F(2) the number of i's, etc.

Then we print asterisks across the page according to the value of F(K) (lines 220 to 250).

```
10  REM★★FREQUENCY DISTRIBUTION★★
20  REM★★PREPARATION PICTURE★★★★★
30  REM★★FREQUENCY-LIST IS F(K)★★
35  DIM F(6)
40  LET K=1
42  PRINT "VOWEL", K
44  PRINT "ENTER FREQUENCY ";
46  INPUT F(K)
48  PRINT F(K)
60  IF F(K)=-9999 THEN GOTO 110
70  LET K=K+1
80  GOTO 42
90  REM★★★★★★★★★★★
100 REM★★DON'T ADD -9999 TO LIST★★★
```

reading the frequencies and storing them in F(1), F(2) . . .

```
110 LET N=K−1
120 REM★★★★★★★★★★
200 REM★★PRINT ROUTINE★★★
205 CLS
210 PRINT
220 FOR X= 1 TO N
230 FOR Y=1 TO F(X)
240 PRINT "★";
250 NEXT Y
260 PRINT
270 PRINT
280 NEXT X
300 REM★★★★★★★★★
```

printing ★ across the page

Program 4 Drawing a frequency distribution

```
RUN

★★★★★★★★★

★★★★★★★★★★★★★★★★

★★★★★★★★★★

★★★★★★★★★★

★★★★★★
```

$\boxed{K}$ Program 4.

Frequency diagram for length of words

If we want to draw a diagram of the frequencies with which the word lengths occurred in SAQ 1, we need to modify Program 4. Two modifications are necessary:

First the frequency list contains more items. There are 15 frequencies (1 to 15) plus −9999, so that's 16 items and we add 35 DIM F(16) to Program 4.

Second the print routine at line 220 will run into problems when the frequency is zero. We can't drive the FOR. . . NEXT loop from 1 to 0! So we must prevent the program going into the FOR. . . NEXT loop when the frequency is zero. To do this we add 225 IF F(X)=0 THEN 260.

We must also modify the input instructions to:

```
42   PRINT "LENGTH",K
44   PRINT "ENTER FREQUENCY ";
```

So the program is

Program 4

insert 35 DIM F(16)
insert 42 PRINT "LENGTH",K
insert 44 PRINT "ENTER FREQUENCY ";
insert 225 IF F(X)=0 THEN 260

A run of the modified Program 4 produces:

```
★
★★★★★
★★★★
★★★★★★
★★★★★★★★★

★
★★★
★

★
```

——————————→——————print out ends about here!

Figure 2 Frequency diagram of modified Program 4

[K] Program 4 (modified).

5.5 Tabulation

We've got the essential ingredients of a picture, but it is still far from being a meaningful diagram. It will help if we have the facility to move the print head across the page or screen to any pre-determined position. In typing this is called tabulation (to arrange in tabular or table-form). In BASIC the TAB function does this for us.

We take the same approach as we did in Unit 3, namely to write a snippet of program which explains itself – an approach well worth cultivating!

First, look at what happens if you number print positions across the screen:

```
50  PRINT"12345678901234567890123456789012
60  PRINT"A";TAB(5);"E";TAB(7);"I";TAB(19);"O";TAB(31);"U"

RUN
12345678901234567890123456789012
A    E I           O           U
```

Program 5

You can see that TAB(5) printed E at the sixth position. Why? Because the machine counts print positions from position 0. This is demonstrated by Program 6 where the scale across the screen goes from 0:

```
50   PRINT"01234567890123456789012345678901
60   PRINT"A";TAB(5);"E";TAB(7);"I";TAB(19);"O";TAB(31);"U"

RUN
01234567890123456789012345678901
A    E I           O           U
```

Program 6

Now TAB(5) goes to the position labelled 5 but it is still in the sixth position across the screen.

Note **On the ZX81 TABA is sufficient to TAB to position A but many computers need brackets, i.e. TAB(A). We have therefore put brackets in all our TAB statements.**

SAQ 5

Write a program to print COL 1, COL 2, COL 3 across the screen with COL 1 starting at position 0, COL 2 at position 10 and COL 3 at position 20.

Variable TAB and its effects

We can drive line 60 of the vowel print with a FOR. . . NEXT loop to produce an actual table.

```
50   FOR I=1 TO 7
60   PRINT"A";TAB(5);"E";TAB(7);"I";TAB(19);"O";TAB(31);"U"
70   NEXT I
```

Program 7

```
RUN
A    E I           O           U
A    E I           O           U
A    E I           O           U
A    E I           O           U
A    E I           O           U
A    E I           O           U
A    E I           O           U
```

Here is another example which shows how we can drive TAB with a variable. If we use TAB(V) where V is a variable, we can drive the print head to different positions across the screen. The program

```
30   FOR A=1 TO 10
40   PRINT TAB(A);"HELLO"
50   NEXT A
60   STOP
```

Value of A in TAB(A) is determined by the loop variable A.

Program 8

produces:

```
RUN
HELLO
```

```
HELLO
 HELLO
  HELLO
   HELLO
    HELLO
     HELLO
      HELLO
       HELLO
        HELLO
```

We can go one step further and combine these two effects in one program:

```
50  FOR I=1 TO 7
60  PRINT TAB(0+I);"A";TAB(5+I);"E";TAB(7+I);"I";
65  PRINT TAB(19+I);"O";TAB(31+I);"U"
70  NEXT I
80  STOP
```

Program 9

```
RUN
A    E I         O          U
 A    E I         O          U
  A    E I         O          U
   A    E I         O          U
    A    E I         O          U
     A    E I         O          U
      A    E I         O          U
```

SAQ 6
Write a program segment to input three numbers of the user's choice which will place the string "HEADING" at three different positions across the same output line.

TAB and the frequency diagram
We are now in a position to set out the frequency diagram of figure 2 in a more attractive manner.

The print routine of the modified Program 4 (lines 200 to 300) was:

```
200  REM★★PRINT ROUTINE★★★★★★★★★★★★★★
210  PRINT
220  FOR X=1 TO N
225  IF F(X)=0 THEN GOTO 260
230  FOR Y=1 TO F(X)
240  PRINT "★";
250  NEXT Y
260  PRINT
270  PRINT
280  NEXT X
300  REM ★★★
```

Program 4 (modified)

We add:
line 212 to print column headings.

line 214 to print a rule across the screen.

line 216 to start the column divides (the rest of the divides we printed by the following loop).

line 222 (in the loop) prints X and F(X) across the page plus the column divides. This line ends in ";" which makes the next PRINT instruction (line 240) appear on the same line.

You will probably have to study this carefully to see all the detail in it:

```
10 to 120 see Program 4.
200   REM★★PRINT ROUTINE★★★★★
210   PRINT
212   PRINT"LENGTH";TAB(8);"FREQ";TAB(18);"TALLY"
214   PRINT"------------------------"
216   PRINT TAB(7);"I";TAB(12);"I"
220   FOR X= 1 TO N
222   PRINT TAB(2);X;TAB(7);"I";TAB(9);F(X);TAB(12);"I";TAB(14);
225   IF F(X)=0 THEN GOTO 260
230   FOR Y=1 TO F(X)
240   PRINT"★";
250   NEXT Y
260   PRINT
280   NEXT X
300   REM★★★★★
```

Program 10

```
RUN
LENGTH    FREQ      TALLY                      212
------------------                             214
          I    I                               216
1         I  1 I    ★
2         I  5 I    ★★★★★
3         I  4 I    ★★★★
4         I  6 I    ★★★★★★
5         I  9 I    ★★★★★★★★★
6         I  0 I
7         I  1 I    ★
8         I  3 I    ★★★
9         I  1 I    ★
10        I  0 I
11        I  0 I
12        I  1 I    ★
13        I  0 I
14        I  0 I       230–250 as before but using
15        I  0 I       TAB(14) as a base line (from line 222)

     effect of line 222
```

K Program 10

Exercise 2
Modify Program 4 to give a print-out similar to that developed for Program 10 and to include the following points:

(a) an appropriate change of headings;
(b) a print-out of the letters A, E, I, O and U as appropriate in the left-hand column;
(c) an appropriate scale at the base of the diagram.

5.6 Cutting up strings

Let's now look at a string which, though being an entity in its own right, contains more than one item of information. For example, 23 June 1971 is a single date but there are occasions when we only want to look at part of it, e.g. the month.

Filing dates
How many times have you been faced with a box on a form like this?

DATE						
	D	D	M	M	Y	Y

If we look at D D M M Y Y the presentation has problems. Compare

23rd June 1971, or 230671

and 14th Sept 1973, and 140973.

The later date has the smaller number. Whereas with

4 July 1933, or 040733

15 Jan 1967, and 150167

the later date has the larger number. Clearly then D D M M Y Y is not very useful for filing dates.

The solution is to put the dates in the form Y Y M M D D. This makes the four dates above:

330704, 670115, 710623, and 730914

giving date and number consistency.

Dates are usually stored as numbers in the machine for use in calculations but are entered as strings to allow checking procedures to occur before they are stored.

If we are interested in a salary increment, then the year and month parts of the number would be important. If we are a music centre and send out reminders to our clients every three months to have their pianos tuned, then only the month may be important. The whole data-string is important in its own right, but we can see that there may be valid reasons for cutting it up.

X$(A TO B)
If we want to consider part of a string, then we need a BASIC statement that will do this for us. On the ZX81, the statement is

X$ (A TO B).

This gives the characters of X$ starting at position A and going on to position B. Thus,

if X$ = "CUTTING"
then X$(3 TO 5) = TTI

You can check how this works with the following program.

```
10   REM★★SLICING DEMONSTRATION★★
12   PRINT "ENTER WORD TO BE SLICED"
14   INPUT M$
16   PRINT M$
20   PRINT "INPUT START OF SLICE"
30   INPUT A
40   PRINT A
50   PRINT "INPUT END OF SLICE"
60   INPUT B
70   PRINT B
80   LET N$ = M$(A TO B)
90   PRINT "M$(";A;" TO ";B;")=";N$
100  STOP
```

Program 11

[K] Program 11.

SAQ 7
If A$ = 1A2B3C4D, what are the following:

(a) A$(3 TO 5)
(b) A$(1 TO 4)
(c) A$(4 TO 4)

LEFT$, RIGHT$ AND MID$
In other BASICS you will find

LEFT$(X$,I)
RIGHT$(X$,I)
MID$(X$,I, J)

LEFT$(X$,I) gives you the characters 1 to I starting from the left of the string. So

LEFT$(X$,I) = X$(1 TO I)

RIGHT$ (X$,I) gives you the I characters starting I characters from the left end of the string and going on to the end of the string. So

RIGHT$(X$,I) = X$(LENX$−I+1 TO LENX$)

MID$(X$,I,J,) gives you J characters of X$ starting at the I'th character. So

MID$(X$,I,J) = X$(I TO I+J)

Cutting up strings of variable length
The following program produces all possible left-strings for any word you input. Note that the program measures the length of the word you input with LEN(X$).

```
20   PRINT "ENTER A STRING"
30   INPUT X$
35   PRINT X$
40   FOR I=1 TO LEN(X$)
50   LET A$=X$(1 TO I)
60   PRINT A$
70   NEXT I
80   STOP
```

LEN(X$) acts as the upper limit of the loop.

Program 12

```
RUN
ENTER A STRING? HAMSTRING
             1234567890
1            H
2            HA
3            HAM
4            HAMS
5            HAMST
6            HAMSTR
7            HAMSTRI
8            HAMSTRIN
9            HAMSTRING
```

[K] Program 12.

Exercise 3
Write a program which allows you to input words one at a time and which will output those words which begin with a vowel.

Exercise 4
Write a program to change the output of Program 12 to:

```
     F
    EF
   DEF
  CDEF
 BCDEF
ABCDEF
```

SAQ 8
Write a program to accept as input London telephone numbers in the form 01 XXX XXXX and output the exchange codes only. (Remember that the spaces are characters just as much as the digits.)

Mid-string program

As you have probably spotted, both MID$ and X$(A TO B) can cut left sub-strings and right sub-strings if we want them to. In other words, they can give us every possible sub-string. Here is a program that does this. First it prints out all sub-strings of length 1, then all of length 2 and so on until it prints the whole word which is the only sub-string of the same length as the word itself!

```
10  REM★★STRING TEST★★
20  PRINT "ENTER A STRING"
21  INPUT X$
22  PRINT X$
25  FOR J=1 TO LEN(X$)
26  CLS
27  PRINT X$; TAB(10);J;"LETTERS"
28  PRINT
30  PRINT "J";TAB(5);"I";TAB(10);"1234567890"
40  FOR I=1 TO (LEN(X$)-J+1)
50  LET A$=X$(I TO I+J-1)
60  PRINT" ";J;TAB(6);I;TAB(10);A$
70  NEXT I
71  PAUSE 100
72  POKE 16437,255
75  NEXT J
80  STOP
```

Make the computer pause before showing the next table

Program 13

```
RUN
ENTER A STRING? STRING
```

Note headings and scale

```
J        I    1 2 3 4 5 6 7 8 9 0
1        1    S
1        2    T
1        3    R
1        4    I
1        5    N
1        6    G

2        1    S T
2        2    T R
2        3    R I
2        4    I N
2        5    N G

3        1    S T R
3        2    T R I
3        3    R I N
3        4    I N G

4        1    S T R I
4        2    T R I N
4        3    R I N G

5        1    S T R I N
5        2    T R I N G

6        1    S T R I N G
```

[K] Program 13.

5.7 VAL

Having found a method of cutting up strings, we now need a method of examining what we have got. One such method is to use VAL(A$) which looks at the numeric value of A$.

VAL(A$) on the ZX81 gives us the numerical value of the string A$ provided A$ only contains numbers.

Program to demonstrate VAL

The following program gives VAL(N$) for any string you input.

```
10  REM★★THE VAL FUNCTION★★
20  PRINT "NEXT STRING"
21  INPUT N$
22  PRINT N$
25  IF N$="ZZZZ" THEN GOTO 999
30  LET N=VAL(N$)
60  PRINT" "
70  PRINT N
80  PRINT" "
90  GOTO 20
999 STOP
```

Program 14

```
NEXT STRING
456

456

NEXT STRING
789

789

NEXT STRING
56T
```

c/30 —— Note error message. The ZX81 cannot calculate VAL(56T) since 56T is only partly numeric.

[K] Program 14.

SAQ 9

What are the values of the following?

(a) VAL (A$) where A$=54
(b) VAL(B$) where B$=76XY
(c) VAL(C$) where C$=A3
(d) VAL (D$) where D$ = −132
(e) VAL(E$(1 TO 2)) where E$=593
(f) VAL(F$(1 TO 1)) where F$=8AM
(g) VAL(G$(1 TO 2)) where G$=Z35
(h) VAL(H$(3 TO 3)) where H$=593
(i) VAL(I$(2 TO 3)) where I$=8AM
(j) VAL(J$(2 TO 3)) where J$=Z35

Date string check

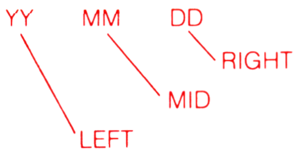
We are now in a position to see how VAL can be put to practical use, in this case to check the accuracy of dates keyed into a computer. This is typical of what happens in computers all the time. We know errors will frequently happen when data is keyed in so, wherever possible, we try to use the computer to detect the errors.

Example 5

Write a program to carry out data checks on the 3 fields of a 6-digit date-string.

Solution

The date-string D$ has three fields in the form:



We are going to consider the years 1980 and 1981 only; so:

VAL(D$(1 TO 2)) should have a range of 80–81,
VAL(D$(3 TO 4)) should have a range of 1–12,
VAL(D$(5 TO 6)) should have a range of 1–31.

This is a fairly complex process as we first need to decide on the steps involved. These are given in the following flowchart (Figure 3).

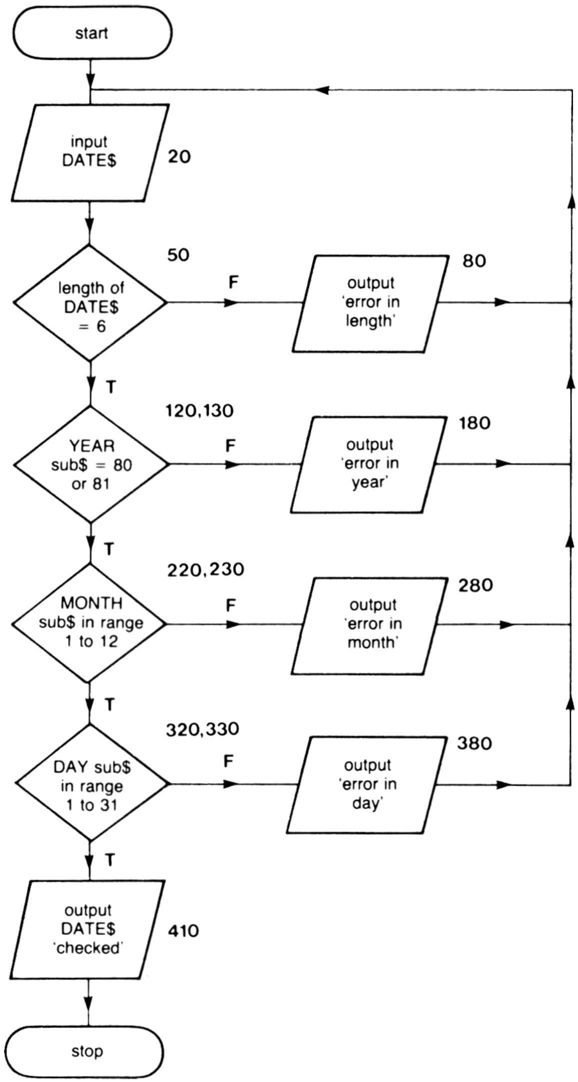


Figure 3 Flowchart showing the four checks on YYMMDD

The program is fairly straightforward as it goes through each of the four checks. If any check fails, the program prints an error message and returns control to line 20. If there are no errors it runs through to line 410 where the correct entry is confirmed.

```
10  REM★★DATE CHECK★★
20  PRINT "NEXT DATE"
21  INPUT D$
22  PRINT D$                                         length check
30  IF D$="ZZZZ" THEN GOTO 900
50  IF LEN(D$)=6 THEN GOTO 110
80  PRINT"!!!!!!ERROR IN DATE LENGTH!!!!!!"          included here, for demon-
90  GOTO 20                                          stration purposes; would
100 REM★★★★★★★★                                      not appear in a checking
110 PRINT"STRING LENGTH CORRECT"                     routine normally.
120 IF VAL(D$(1 TO 2))=80 THEN GOTO 210
130 IF VAL(D$(1 TO 2))=81 THEN GOTO 210              year check
180 PRINT"!!!!!!ERROR IN YEAR FIELD!!!!!!"
190 GOTO 20
200 REM★★★★★★★★
210 PRINT"YEAR FIELD CORRECT"                        demonstration only
220 IF VAL(D$(3 TO 4))<1 THEN GOTO 280
230 IF VAL(D$(3 TO 4))<=12 THEN GOTO 310             month check
280 PRINT"!!!!!!ERROR IN MONTH FIELD!!!!!!"
290 GOTO 20
300 REM★★★★★★★★
310 PRINT"MONTH FIELD CORRECT"                       demonstration only
320 IF VAL(D$(5 TO 6))<1 THEN GOTO 380
330 IF VAL(D$(5 TO 6))<=31 THEN GOTO 410             day check
380 PRINT"!!!!!!ERROR IN DAY FIELD!!!!!!"
390 GOTO 20
400 REM★★★★★★★★★★
410 PRINT"DATE STRING WITHIN CHECK LIMITS"
490 GOTO 20
900 PRINT"END OF DATE CHECK"
```

Program 15

Typical run

```
RUN
NEXT DATE? 1234567
!!!!!!ERROR IN DATE LENGTH!!!!!!
NEXT DATE? 123456
STRING LENGTH CORRECT
!!!!!!ERROR IN YEAR FIELD!!!!!!
NEXT DATE? 803456
STRING LENGTH CORRECT
YEAR FIELD CORRECT
!!!!!!ERROR IN MONTH FIELD!!!!!!
NEXT DATE? 801256
STRING LENGTH CORRECT
YEAR FIELD CORRECT
MONTH FIELD CORRECT
```

```
!!!!!!ERROR IN DAY FIELD!!!!!!
NEXT DATE? 800131
STRING LENGTH CORRECT
YEAR FIELD CORRECT
MONTH FIELD CORRECT
DATE STRING WITHIN CHECK LIMITS
NEXT DATE? ZZZZ
END OF DATE CHECK
```

[K] Program 15.

Assignment 5

1. Write a program to find the frequency with which each vowel occurs in the words in the JUDE data of Example 2, giving also a summary of the total number of vowels and consonants which occur in these words.
2. Write a program to input a string of characters and to output this string in reverse order.

Objectives of Unit 5

Check that you are now able to write simple programs:

Using LEN(A$) ☐

Using LETC(I)=C(I)+1 to count frequencies ☐

To print a frequency diagram ☐

Using TAB to print in columns ☐

Using TAB to print a frequency table with headings and scale ☐

Using X$(A TO B) to act as LEFT$(X$,I) ☐
RIGHT$(X$,I) and MID$(X$,I,J) ☐

Using VAL(A$) ☐

Answers to SAQ's and Exercises

SAQ 1
(a) 3; (b) 1; (c) 2 (not 72 – LEN counts the number of characters); (d) 7 (LEN counts the characters regardless of whether they are numbers, letters or spaces).

SAQ 2

Your answer should be:

Word length	Count	Total
1	1	1
2	~~1111~~	5
3	1111	4
4	~~1111~~ 1	6
5	~~1111~~ 1111	9
6		0
7	1	1
8	111	3
9	1	1
10		0
11		0
12	1	1
13		0
14		0
15		0

SAQ 3

(a) C(4)=1 (Not 3 or 4! C(4) has counted the number of 3's inputted.)
(b) C(10)=2
(c) C(2)=4
(d) C(1)=3

Exercise 1

The solution appears in the following text.

SAQ 4

★★
★★★★★
★★★★★★★
★★★★★★★★
★★★
★

SAQ 5

```
10 PRINT"COL1";TAB(9);"COL2";TAB(19)
```

SAQ 6

```
10  INPUT A,B,C
20  PRINT TAB(A);"HEADING";TAB(B);"HEADING";TAB(C);"HEADING"
```

Program 16

Exercise 2

The program asks you for the vowels one by one and their frequencies. 'Vowel No 6' terminates the input – you can enter any letter you like followed by the frequency −9999.

```
10  REM★★FREQUENCY DISTRIBUTION★★
15  REM★★PREPARATION PICTURE★★★★★
20  REM★★FREQUENCY-LIST IS F(K)★★
25  DIM V$(6,1)
28  DIM F(6)
30  FOR K=1 TO 6
35  PRINT "VOWEL NUMBER ";K;
36  INPUT V$(K)
38  PRINT V$(K)
40  PRINT "FREQUENCY ";K;
43  INPUT F(K)
46  PRINT F(K)
49  IF F(K)= −9999 THEN GOTO 110
50  NEXT K
90  CLS
100 REM★★DON'T ADD −9999 TO LIST★★★
110 LET N=K−1
120 REM★★★★★★★★
200 REM★★PRINT ROUTINE★★★★★★★★★★★★★
210 PRINT
212 PRINT"VOWEL";TAB(8);"FREQ";TAB(18);"TALLY"
214 PRINT"========================="
220 FOR X= 1 TO N
222 PRINT TAB(2);V$(X);TAB(7);"I";TAB(10);F(X);TAB(14);"I";TAB(16);
230 FOR Y=1 TO F(X)                          prints out from word list
240 PRINT "★";
250 NEXT Y
260 PRINT
280 NEXT X
290 PRINT"....SCALE....";TAB(15);"0....5....0....5....0"
300 REM★★★★★★★
```

Program 17

```
VOWEL    FREQ          TALLY
=================================
  A        I     9    I  ★★★★★★★★★
  E        I    16    I  ★★★★★★★★★★★★★★★★
  I        I    10    I  ★★★★★★★★★★
  O        I    10    I  ★★★★★★★★★★
  U        I     6    I  ★★★★★★
....SCALE....     0....5....0....5....0
```

[K] Program 17.

SAQ 7

(a) 2B3; (b) 1A2B; (c) B.

Notice that LEFT$ and RIGHT$ treat all characters in a string in the same way. It doesn't matter whether they are numbers or letters, they still get counted.

Exercise 3

```
10 REM★★IS LEFT-MOST CHARACTER A VOWEL?★★
20 PRINT "NEXT WORD"
25 INPUT W$
28 CLS
30 IF W$="ZZZZ" THEN GOTO 9999
40 LET L$=W$(1 TO I)
50 IF L$="A" THEN GOTO 200
60 IF L$="E" THEN GOTO 200
70 IF L$="I" THEN GOTO 200
80 IF L$="O" THEN GOTO 200
90 IF L$="U" THEN GOTO 200
100 GOTO 20
190 REM★★★★★★★★★★
200 PRINT
210 PRINT L$, W$
220 GOTO 20
230 REM★★★★★★★★★★
9999 STOP
```

Program 18

[K] Program 18.

Exercise 4

```
10 REM★★STRING TEST★★
20 PRINT "ENTER 6-CHARACTER STRING"
25 INPUT X$
30 PRINT TAB(10);"1234567890"
40 FOR I=1 TO 6
50 LET A$=X$(6-I+1 TO 6)
60 PRINT I;TAB(16-I);A$
70 NEXT I
```

Program 19

```
RUN
ENTER 6-CHARACTER STRING? ABCDEF
          1234567890
1
2               F
3              EF
4             DEF
5            CDEF
6           BCDEF
           ABCDEF
```

[K] Program 19.

SAQ 8

```
10 PRINT "NEXT TELEPHONE NUMBER"
15 INPUT N$
18 PRINT N$
20 LET A$=N$(4 TO 6)
```

```
30  PRINT A$
40  GOTO 10
```

Program 20

$\boxed{K}$ Program 20.

SAQ 9

(a) 54; (b) 0; (c) 0; (d) −132; (e) 59; (f) 8; (g) 0 (starts with a letter); (h) 3; (i) 0 (contains A which is a letter); (j) 35.

UNIT 6

Mainly about dice and games

6.1	Random numbers	144
6.2	The RND function	145
6.3	Random number postscript	150
6.4	Two examples	151
6.5	Keeping scores	155
6.6	Concatenation	157
6.7	STR$	158
	Assignment 6	160
	Objectives of Unit 6	160
	Answers to SAQ's and Exercises	161

6.1 Random numbers

The programming function which allows us to inject a sense of fun into a program is the one which generates random numbers. This function is at the heart of many of the game-playing and simulation programs which are now available for microcomputers.

You will have met random numbers when playing games; games which involve tossing a coin or throwing a dice, or drawing numbers out of a hat. These domestic games have become institutionalised in casinos, bingo clubs, the ritualistic draw for the FA Cup competition, and, of course, on a larger scale, the monthly draw for premium bonds. Although we all have an intuitive idea of what we mean by a sequence of random numbers, it is quite difficult to define the idea clearly. Let's have a look at some number sequences to try and clarify this idea.

Here are three 'thought experiments' each of which involves throwing a six-sided die fifteen times. Imagine that in the first experiment the uppermost values of the dice had those values shown in sequence A shown in Figure 1. The second experiment generated the numbers shown in sequence B and the third experiment gave us the numbers shown in sequence C.

Sequence A
5,1,2,4,6,3,2,1,6,3,5,4,3,4,2
Sequence B
6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
Sequence C
1,2,3,4,5,6,1,2,3,4,5,6,1,2,3

Figure 1 Random sequences?

Most of us would be quite happy that sequence A represented a typical sequence of numbers generated by throwing a die fifteen times. This number sequence shows no definable patterns or repetitions and each number occurrence would seem to be 'equally likely'. We are not surprised at the appearance of any of the sub-sequences in this main sequence. By contrast, however, sequence B is quite unreasonable. We would certainly not expect to have thrown fifteen sixes with fifteen consecutive throws of the die. We would be highly suspicious had this happened and we would blame a weighted die. Intuitively, we would be prepared to accept that sequence A had occurred 'by chance' but would not be prepared to accept that this was so for sequence B.

Another feature about random number sequences which we learn by experience is that, in long sequences, localised 'unfair' occurrences iron themselves out. What we mean by this is that after, say, a hundred throws, we would expect on average about sixteen ones, about sixteen twos, sixteen threes and so on. In other words, over a longer sequence we expect the 'laws of chance' to apply. If we now consider sequence C with its emerging pattern ' . . .6,1,2,3,4,5,6,1 . . . ' continued for a hundred throws then this long term averaging-out effect would be satisfied. But once again this sequence would not be intuitively acceptable to us as random because we would not expect this sequential pattern to persist over a hundred throws by chance alone.

These concepts of 'statistical averaging' over a sequence of throws, and of the 'reasonableness' of the patterning of the numbers in sequence are intuitively

acquired from games of chance. There are statistical techniques to test these two features of a random number sequence but we will not be concerned with those techniques here.

A computer is a very determinate machine. You will therefore not be surprised to learn that quite special features have to be programmed into the machine to achieve a sequence of random numbers. For our uses, however, we will assume that a table of random numbers has been stored in the machine's memory. The sequence of numbers is very long and generation would have to occur for a long time before the sequence repetition became apparent. To achieve a different random number sequence from one program execution to another, all that a machine has to do is to start reading this table of random numbers from a different point. This starting point is often referred to as the 'seed' and we talk about random number sequences as starting from different seeds. Because the computer has to 'contrive' random number sequences, the numbers produced are usually referred to as pseudo-random numbers.

6.2 The RND function

If you run the following program, you will be able to see the effect of RND.

```
10  FOR I=1 TO 10
20  LET B=RND
30  PRINT B
40  NEXT I
```

Program 1 RND

A typical run produces numbers like:

```
0.32705689
0.5300293
0.75273132
0.451239
0.13490295
0.11869812
0.90336609
0.75256348
0.4425354
0.19078064
```

K Program 1

These certainly look like random numbers and, if you key RUN, you will get a completely different list. Actually the computer has a list of 65536 jumbled up numbers and RND plunges into that list and reads as many numbers as you ask for.

The fact that the random numbers are available in a fixed sequence means that we can repeat a sequence of random numbers if we want to. On the ZX81 we do this by the keyword RAND. e.g. RAND25 will start reading numbers from the sequence at the 25th number. Try the following program to convince yourself that RAND

does fix the starting point of the sequence.

```
5    RAND7
10   FOR I=1 TO 10
20   LET B=RND
30   PRINT B
40   NEXT I
```

Program 2

🄺 Program 2. Run it several times and note the sequences.

RND

What the above investigation demonstrates is that

RND

will give random numbers within the range 0–1 with 0 included but 1 excluded.

How can we extend the range to get other random numbers? Quite simply by multiplying RND by another number. So:

RND gives a random number in the range 0–1;
6★RND gives a random number in the range 0–6;
and 52★RND gives a random number in the range 0–52;
etc.

You can think of RND as a 'conversion factor' which changes at will. The following program tries out this idea.

```
10   REM★★RND AS A CONVERSION FACTOR★★
30   PRINT "  I";TAB(8);"1★RND";TAB(20);"6★RND"
40   PRINT "---";TAB(8);"------";TAB(20);"------"
50   FOR I=1 TO 10
60   LET B=RND(1)
70   LET C=6★B
90   PRINT I;TAB(8);B;TAB(20);C
100  NEXT I
110  STOP
```

Program 3   RND as a conversion factor

Run with line 70 as LET C=6★B:

```
RUN
I              RND               6★RND
---            ------            ------
1              0.6658783         3.9952698
2              0.94125366        5.647522
3              0.59408569        3.5645142
4              0.55688477        3.3413086
5              0.76686096        4.6011658
6              0.51483154        3.0889893
7              0.61291504        3.6774902
8              0.96907044        5.8144226
9              0.68031311        4.0818787
10             0.23834229        0.14300537
```

Run with line 70 as LET C=52★B plus change of headings in line 30:

RUN

I	RND(1)	52★RND(1)
---	------	----------
1	0.78868103	41.011414
2	0.15130615	7.8679199
3	0.34892273	18.143982
4	0.16993713	8.836731
5	0.74623108	38.804016
6	0.96762085	50.316284
7	0.57159424	29.7299
8	0.87005615	45.24292
9	0.25434876	13.226135
10	0.07699585	4.0037842

[K] Program 3.

SAQ 1
Write a program to print out 6 random numbers in the range 0 to 5.999999.

The RND+1 function
If you look again at the output of Program 3 on the run with 6★RND, the numbers were:

3.9952698
5.647522
3.5645142
3.3413086
4.6011658
3.0889893
3.6774902
5.8144226
4.0818787
0.14300537

Look now at the numbers before the decimal point (picked out in colour). They are:

3,5,3,3,4,3,3,5,4,0

i.e. members of the set

(0,1,2,3,4,5)

But, if we were throwing dice we would generate members of the set (1,2,3,4,5,6). All we have to do then, is to add 1 to each member of the first set to get the second.

Now in games we frequently want to throw a dice (outcomes 1,2,3,4,5,6) or use a pack of cards (52 outcomes) so we are particularly interested in the functions:

6★RND+1 and 52★RND+1

The following program allows us to explore 6★RND.

```
10   REM★★RND+1 AS A CONVERSION FACTOR★★
30   PRINT "  I";TAB(8);"1★RND";TAB(20);"6★RND+1"
40   PRINT"--- ";TAB(8);"------ ";TAB(20);"------- "
50   FOR I=1 TO 10
60   LET B=RND
70   LET C=6★B+1                 Note +1
90   PRINT I;TAB(8);B;TAB(20);C
100  NEXT I
110  STOP
```

Program 4 :6★RND+1

```
RUN
I           1★RND          6★RND+1
---         ------         ---------
1           0.89263916     6.355835
2           0.94805908     6.6883545
3           0.10447693     1.6268616
4           0.83679199     6.020752
5           0.75958252     5.5574951
6           0.96896362     6.8137817
7           0.67230225     5.0338135
8           0.42303467     3.538208
9           0.72825623     5.3695374
10          0.6195221      4.7171326
```

[K] Program 4.

The INT function

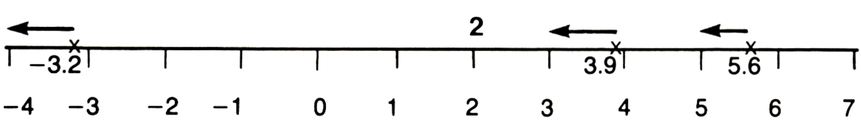
If you now look at the columns in the runs of Program 4 picked out in colour you will now see that we have generated the random numbers we needed. Column 3 has numbers from 1 to 6 and column 4 has numbers from 1 to 52.

But what about all the garbage to the right of the decimal point? Well, we have a function to get rid of that: the INT function.

The effect of INT(X) is to give the whole number (or integer part) of the number X, i.e. the largest integer which is not larger than X. The effect of INT is to 'chop' down to the next highest whole number:

INT(5.6) = 5
INT(3.9) = 3
INT(−3.2) = −4
INT(2) = 2

If INT(−3.2) = −4 surprises you, look at the number line and remember that INT always chops down to the next whole number. It doesn't 'round' numbers.



SAQ 2

What are the values of the following:

(a) INT(4.5)
(b) INT(9.1)
(c) INT(−2.5)
(d) INT(−0.99)
(e) INT(1.01)

Now, with INT, we can at last generate the whole numbers from 1 to 6 and from 1 to 52 to use as dice or cards. All we need is

INT(6★RND +1)
and INT(52★RND +1)

The following program prints out the values of these two functions:

```
10  REM★★THE INT FUNCTION★★
30  PRINT "  I";TAB(5);"RND★1";TAB(12);"INT(6★ – – –";TAB(22);
    "INT(52★ – – –"
35  PRINT TAB(12);"– – – RND+1)";TAB(22);"– – – RND+1"
40  PRINT "– – –";TAB(5);"– – – – – –";TAB(12);– – – – – – – – – –";TAB(22);
    "– – – – – – – – – –"
45  PRINT
50  FOR I=1 TO 10
60  LET B=RND
70  LET C=INT(6★B+1)
80  LET D=INT(52★B+1)
90  PRINT;TAB(4);B;TAB(18);C;TAB(25);D
100 NEXT I
110 STOP
```

Program 5   INT to give whole numbers

```
RUN
                                INT(6★– – –          INT(52★– – –
I       RND(1)                  – – – RND+1)         – – – RND+1)
– – –   – – – – – –             – – – – – – – – – –  – – – – – – – – – –
1       0.48942566              3                    26
2       0.70750427              5                    37
3       0.063140869             1                    4
4       0.7366333               5                    39
5       0.24778748              2                    13
6       0.58491516              4                    31
7       0.86911011              6                    46
8       0.18339539              2                    10
9       0.75558472              5                    40
10      0.66912842              5                    35
```

[K] Program 5.

6.3 Random number postscript

The following program simulates the tossing of a die 100 times.

```
10  REM★★100 TOSSES OF A DIE★★
30  FOR I=1 TO 10
40  FOR J=1 TO 10
50  LET X=INT(6★RND+1)
60  PRINT X;" ";
70  NEXT J
80  PRINT
90  NEXT I
100 STOP
```

Program 8 Die toss

```
RUN
4  1  6  6  4  3  2  4  3  1
6  6  2  2  5  3  6  5  2  2
3  3  6  3  4  2  6  1  2  6
6  1  4  3  5  2  2  2  1  6
6  3  1  1  6  3  4  5  2  5
6  1  4  2  2  4  5  1  2  4
6  6  5  4  3  4  5  5  1  5
3  5  4  3  5  3  6  6  6  6
1  5  3  4  3  1  1  4  2  1
6  3  6  1  6  1  6  5  3  1
```

[K] Program 8.

Just to make sure that you understand the INT function, try the following questions.

SAQ 3
The program:

```
10  REM★★SAQ★★
20  FOR X=-3.8 TO -1.8 STEP(.2)
30  LET Y=INT(X)
40  PRINT X,Y
50  NEXT X
60  STOP
```

Program 9

prints out 10 pairs of numbers. What are they?

SAQ 4
The program:

```
10  REM★★SAQ★★
20  FOR X=1.6 TO 3.4 STEP(.2)
30  LET Y=INT(X)
40  PRINT X,Y
50  NEXT X
60  STOP
```

Program 10

prints out 9 pairs of numbers. What are they?

6.4 Two examples

This section is made up of two lengthy examples. We suggest that you try and treat them as exercises first and then compare your solution with ours.

Example 1
Write a program to simulate tossing a coin 100 times. Count and output the number of times the coin falls heads and tails.

Solution
The heart of the solution is a random number generator which produces a 1 or a 2.

We will use a 1 to represent a tail and 2 to represent a head. Using this approach, a descriptive algorithm for the solution to the problem is:

1. Start.
2. Set heads total and tails total to zero.
3. Start loop counter.
4. Generate the two values 1 and 2 randomly.
5. If random number equals 2 then go to statement 8 otherwise go on to statement 6.
6. Add 1 to tails total.
7. Go to statement 9.
8. Add 1 to heads total.
9. Add 1 to loop counter.
10. If loop counter <=100 then go to statement 4 otherwise go on to statement 11.
11. Output total heads and total tails.
12. Stop.

Figure 2 Descriptive solution to coin toss

Alternatively you may prefer a flowchart description to the solution:

GLOSSARY

H . . . 'heads' counter
T . . . 'tails' counter
I . . . '100-toss' loopcounter
Y . . . contains each random
Number as generated.

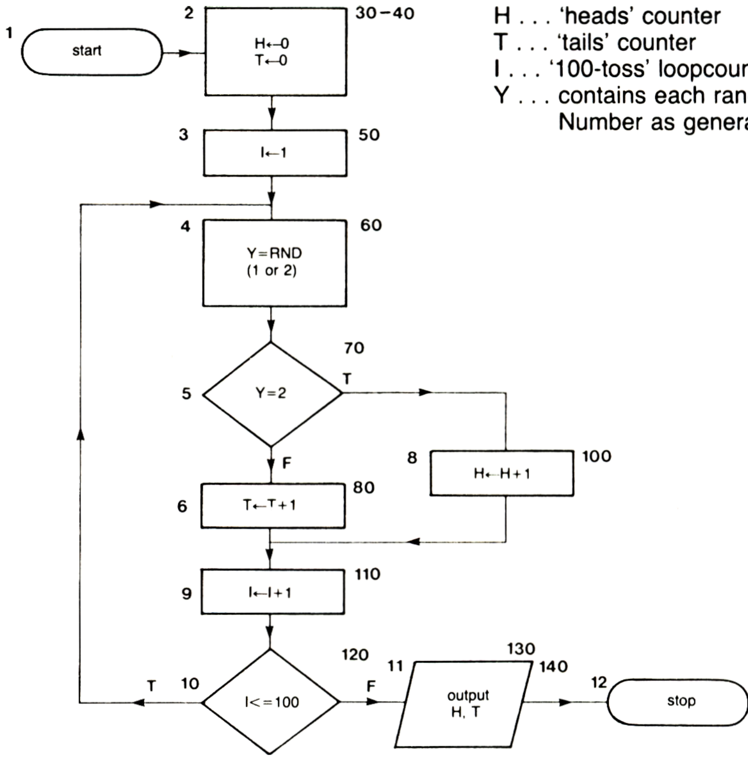


Figure 3 Flowchart for coin toss

And, finally, the program:

```
10  REM★★TOSS 1 COIN 100 TIMES★★
30  LET H=0
40  LET T=0
50  LET I=1
60  LET Y=INT(2★RND(1)+1)
70  IF Y=2 THEN GOTO 100
80  LET T=T+1
90  GOTO 110
100 LET H=H+1
110 LET I=I+1
120 IF I<=100 THEN GOTO 60
130 PRINT"HEADS","TAILS"
140 PRINT H,T
150 STOP
```

Program 11 Coin toss

RUN	
HEADS	TAILS
51	49
RUN	
HEADS	TAILS
57	43
RUN	
HEADS	TAILS
55	45

[K] Program 11.

Example 2
Write a program to simulate tossing 2 coins 100 times. Count and output the number of times that the outcome of this imaginary experiment is: head-head (HH), tail-tail (TT), and head-tail or tail-head (HT or TH).

Solution
We use the same scoring rules: 1 for a tail and 2 for a head but we are now tossing 2 coins. We store the score from the first coin in C1 and the score from the second coin in C2. Then we add C1 and C2 to give the total score for that throw:

S=C1+C2

S can be 2, 3 or 4:

outcome	score
TT	1+1=2
TH or HT	1+2=2+1=3
HH	2+2=4

Then we count how many 2's we get, how many 3's and how many 4's.

Counter for 2's	T2	
Counter for 3's	M1	(M for Mix of H's and T's)
Counter for 4's	H2	

The flowchart of the solution is:

GLOSSARY OF SYMBOLS
H2 . . . total for HH
T2 . . . total for TT
M1 . . . total for HT or TH
C1 . . . random 1 or 2 for coin 1
C2 . . . random 1 or 2 for coin 2
S . . . sum of coin scores.

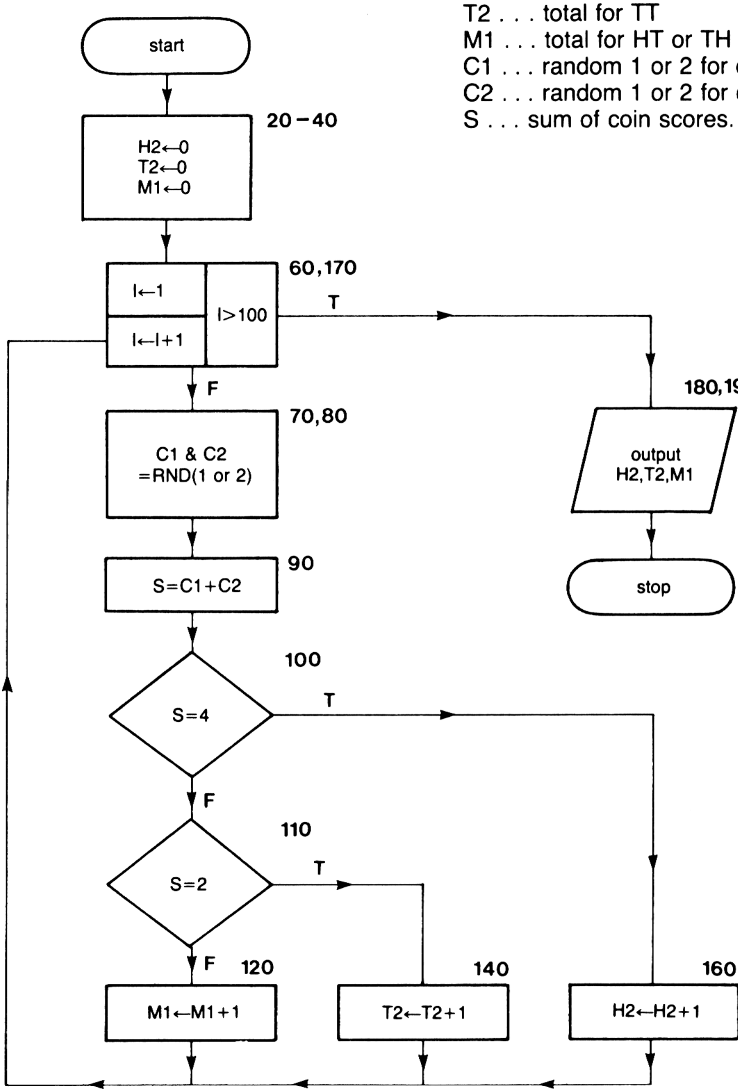


Figure 4 Flowchart for two coin toss

And the program is:

```
10  REM★★TOSS 2 COINS 100 TIMES★★
20  LET H2=0
```

```
30  LET T2=0
40  LET M1=0
60  FOR I=1 TO 100
70  LET C1=INT(2★RND +1)
80  LET C2=INT(2★RND +1)
90  LET S=C1+C2
100 IF S=4 THEN GOTO 160
110 IF S=2 THEN GOTO 140
120 LET M1=M1+1
130 GOTO 170
140 LET T2=T2+1
150 GOTO 170
160 LET H2=H2+1
170 NEXT I
180 PRINT "TT";TAB(10);"HT";TAB(20);"HH"
190 PRINT T2;TAB(10);M1;TAB(20);H2
200 STOP
```

Program 12   Two coin toss

```
RUN
TT      HT      HH
21      54      25

RUN
TT      HT      HH
26      48      26

RUN
TT      HT      HH
26      48      26

RUN
TT      HT      HH
24      55      21
```

[K] Program 12.

6.5 Keeping scores

You may have noticed that we have used some ungainly variable names, such as T2, H2 and M1. Perhaps you have been thinking, 'What about lists? Couldn't they be as useful here as they were with frequency tables?' Indeed they could so let's try a score list S(I) for the coin tossing. So we say:

If the score is 2, add 1 to the number in S(2)
If the score is 3, add 1 to the number in S(3)
If the score is 4, add 1 to the number in S(4)

generally

If the score is S add 1 to the number in S(S)

Application to tossing two coins
If we go back to Example 2, we can re-use lines 10 to 90 and then put in our new scoring system:

```
100 LET S(S)=S(S)+1
```

The program then becomes:

```
10  REM★★TOSS 2 COINS 100 TIMES★★
15  DIM S(4)
20  LET S(4)=0
30  LET S(3)=0
40  LET S(2)=0
60  FOR I=1 TO 100
70  LET C1=INT(2★RND+1)
80  LET C2=INT(2★RND+1)
90  LET S=C1+C2
100 LET S(S)=S(S)+1
110 NEXT I
120 PRINT "TT";TAB(10);"HT";TAB(20);"HH"
130 PRINT S(2);TAB(10);S(3);TAB(20);S(4)
140 STOP
```

Note the addition of 15 DIM S(4)

Program 13

```
RUN
TT          HT          HH
21          58          21

RUN
TT          HT          HH
27          47          26

RUN
TT          HT          HH
31          52          17

RUN
TT          HT          HH
19          55          26
```

[K] Program 13.

Score lists for dice
The score-list for throwing one die would be:

S(1), S(2), S(3) . . . S(6);

and for throwing two dice:

S(2), S(3), S(4) . . . S(12).

Exercise 1
Write a program to simulate throwing a die 100 times. Count and output the number of times each score occurs.

Exercise 2
Modify the program written for Exercise 1 to simulate the throwing of two dice 100

times.

Exercise 3
Write a program to display the data obtained from the program in Exercise 2, in the form of a frequency diagram.

6.6 Concatenation

Having gone to a lot of trouble in Unit 5 to cut up strings, we will now spend some time putting them back together. This second ugliest word in the computing repertoire means to chain or link together. Program 14 shows us what is happening.

```
19  PRINT "FIRST STRING"
20  INPUT A$
21  PRINT A$
29  PRINT "SECOND STRING"
30  INPUT B$
31  PRINT B$
40  PRINT A$+B$
50  STOP
```

Program 14 Concatenation

```
RUN
GET
TOGETHER
GETTOGETHER
```

```
RUN
CONCATE
NATION
CONCATENATION
```

SAQ 5
Write a program to input a word and output its plural assuming that all words only need s adding to make their plurals.

Program 15 shows how we can use concatenation to build up a string from a list of symbols. We have stored the letters in A$ in line 30; in the loop 110–140 we add a new letter to the string on each around the loop.

```
10  REM★★CONCATENATION★★
20  REM★★SET UP DIRECTORY★★
30  LET A$="ABCDEFGHIJ"
50  REM★★★★★
100 C$=""
105 REM★EMPTY C$★★
110 FOR J=1 TO 10
120 C$=C$+A$(J)
130 PRINT J,C$
140 NEXT J
150 STOP
```

Program 15

```
RUN
1              A
2              AB
3              ABC
4              ABCD
5              ABCDE
6              ABCDEF
7              ABCDEFG
8              ABCDEFGH
9              ABCDEFGHI
10             ABCDEFGHIJ
```

Ⓚ **Program 15.**

This process is of great value in textual analysis, but we will use it for codes and games.

6.7 STR$

This function has the reverse effect to the VAL-function. The VAL-function gives the numerical value of a string, and the STR$-function turns a number into just a string of characters.

STR$(X) gives the string representation of the value of X.

Printing STR$
STR$(N) looks very much like N itself as the following program shows:

```
10  REM★★THE STR$-FUNCTION★★
20  PRINT "NEXT NUMBER? ";N
21  INPUT N
22  PRINT N
25  PRINT"01234567890123456789O"
30  PRINT N, STR$(N)
40  STOP
```

Program 16

```
RUN
NEXT NUMBER? 17
01234567890123456789O
 17             17
RUN
NEXT NUMBER?-17
01234567890123456789O
-17             -17
RUN
NEXT NUMBER? 99.34
01234567890123456789O
 99.34          99.34
RUN
NEXT NUMBER?-99.34
01234567890123456789O
-99.34          -99.34
```

[K] Program 16.

However, in each run the second figure (coloured) is treated as a string.

In the next program (Program 17) we make use of the fact that STR$(8) treats 8, say, as a string so that we add the character 8 (as opposed to its value) onto the end of a string.

```
50  REM★★★★★★★★★★★★★★★★★★★★★★
100 C$="":REM★★EMPTY C$★★
105 REM★★EMPTY C$★★
110 FOR J=1 TO 10
120 LET C$=C$+STR$(J)
130 PRINT J,C$
140 NEXT J
150 STOP
```

Program 17

Output on some microcomputers:

```
RUN
1             1
2             1 2
3             1 2 3
4             1 2 3 4
5             1 2 3 4 5
6             1 2 3 4 5 6
7             1 2 3 4 5 6 7
8             1 2 3 4 5 6 7 8
9             1 2 3 4 5 6 7 8 9
10            1 2 3 4 5 6 7 8 9 10
```

Output on ZX81

```
RUN
1             1
2             12
3             123
4             1234
5             12345
6             123456
7             1234567
8             12345678
9             123456789
10            12345678910
```

Thus on the ZX81 we can link the characters in adjacent positions.

[K] Program 17.

Exercise 4
Write a program to input a word from the keyboard, to code each letter as a number and output the code as sequence of numbers.

Guidance if required: set up a directory-list as in Program 17 but for the whole alphabet. Remember the DIM statement. Take each letter of the word and compare it with the items of the directory-list. When found in the directory, add that index, in string form, to the code string.

Exercise 5
Write a program to generate 20 random 3-letter words. (It's interesting to see how many times you have to run this program until you generate a bona fide word.)

Assignment 6

1. Write a program to deal a hand of cards, the size of which is left to you. Print out the hand in the form

   Remember that when a card has been dealt it cannot be dealt again.

   Guidance if required: write the program in sections:
   - set up the deck (we have previously called it a directory);
   - deal (using the RND generator 1–52 is easiest);
   - output.

   When a card has been dealt, put a marker (e.g. an ★) in that position to signal that it cannot be used again.

2. Write a program to simulate a game of snakes and ladders using a 4×4 board and one die.

   Guidance if required: though the board is square it can be represented in memory by a list B(I):
   i.e. B(1), B(2), B(3) . . . B(16)
   B(3)=+4 could be a ladder going up 4 places.
   B(9)=−7 could be a snake going down 7 places, etc.

   Don't forget that your last throw has to give the right number to complete the board at exactly 16.

   Suggestions (not part of the Assignment for your tutor): play the game a few times and estimate the average number of throws needed to run the board. Change the layout of the board and try some more runs. Design a bigger board, etc.

Objectives of Unit 6

When you have finished this unit, check that you are able to: ☐

Use RND to generate random numbers between 0 and 1. ☐

Use INT and RND to generate integer random numbers between 0 and a given integer N. ☐

Simulate the tosses of a coin. ☐

Simulate the tosses of two coins. ☐

Simulate the throw of a die. ☐

Simulate the throws of two dice. ☐

Use score lists. ☐

Concatenate strings. ☐

Use STR$(X). ☐

Answers to SAQ's and Exercises

SAQ 1

```
20  FOR I=1 TO 6
30  LET N=6*RND
40  PRINT N
50  STOP
```

Program 18

SAQ 2

(a) 4; (b) 9; (c) −3; (d) −1; (e) 1.

SAQ 3

```
RUN
-3.8          -4
-3.6          -4
-3.4          -4
-3.2          -4
-3            -3
-2.8          -3
-2.6          -3
-2.4          -3
-2.2          -3
-2            -2
```

SAQ 4

```
RUN
1.6           1
1.8           1
2             2
2.2           2
2.4           2
2.6           2
2.8           2
3             3
3.2           3
```

Exercise 1

Descriptive algorithm for 1 die toss

1. Start.
2. Set the 6 total store locations to zero.
3. Start the 100-throws loop.
4. Generate a random score from the set (1,2,3,4,5,6).
5. Increment the total linked with this score.
6. If loop counter <=100 then go to statement 4 otherwise go on to statement 7.
7. Output score and total for each score value.
8. Stop.

```
10  REM★★THROW 1 DIE 100 TIMES★★
15  DIM S(6)
20  FOR J=1 TO 6
30  LET S(J)=0
40  NEXT J
60  FOR I=1 TO 100
70  LET S=INT(6★RND+1)
80  LET S(S)=S(S)+1
90  NEXT I
100 PRINT
110 PRINT"SCORE","FREQUENCY"
120 FOR K=1 TO 6
130 PRINT K,S(K)
140 NEXT K
150 STOP
```

Program 19

```
RUN
SCORE                 FREQUENCY
1                     12
2                     20
3                     16
4                     18
5                     16
6                     18
```

```
RUN

SCORE                 FREQUENCY
1                     9
2                     20
3                     19
4                     18
5                     14
6                     20
```

RUN

SCORE	FREQUENCY
1	14
2	28
3	17
4	18
5	15
6	8

RUN

SCORE	FREQUENCY
1	15
2	19
3	18
4	17
5	17
6	14

K **Program 19.**

Exercise 2

```
10  REM★★THROW 2 DICE 100 TIMES★★
20  DIM S(15)
30  FOR J=2 TO 12
40  LET S(J)=0
50  NEXT J
70  FOR I=1 TO 100
80  LET S1=INT(6★RND +1)
90  LET S2=INT(6★RND +1)
100 LET S=S1+S2
110 LET S(S)=S(S)+1
120 NEXT I
130 PRINT
140 PRINT"SCORE","FREQUENCY"
150 FOR K=2 TO 12
160 PRINT K,S(K)
170 NEXT K
180 STOP
```

Program 20

RUN

SCORE	FREQUENCY
2	1
3	5
4	10
5	13
6	11
7	19

8	16
9	5
10	10
11	5
12	5

[K] **Program 20.**

How about 1000 throws? Well, just change 70 to 70 FOR I=1 TO 1000 (forget what the REM line says!) and you will get a run such as:

RUN

SCORE	FREQUENCY
2	31
3	59
4	74
5	118
6	151
7	173
8	118
9	100
10	89
11	61
12	26

Suggestions for further programs

1. How will you cope with printing a frequency diagram for this data, as 151 and 173 . . . would 'want to print' off the end of the line?
2. We need a general 'scaling' routine which adjusts to differing line widths, but makes best use of the full width of the page or screen.

Exercise 3

```
10  REM★★FREQUENCY DIAGRAM★★
20  DIM S(15)
30  FOR J=2 TO 12
40  LET S(J)=0
50  NEXT J
70  FOR I=1 TO 100
80  LET S1=INT(6★RND+1)
90  LET S2=INT(6★RND+1)
100 LET S=S1+S2
110 LET S(S)=S(S)+1
120 NEXT I
130 PRINT
140 PRINT"FREQUENCY DIAGRAM"
150 PRINT
160 FOR K=2 TO 12
170 PRINT K;TAB(5);S(K);TAB(10);
175 IF S(K)=0 THEN GOTO 210
```

```
180 FOR L=1 TO S(K)
190 PRINT"★";
200 NEXT L
210 PRINT
220 NEXT K
230 STOP
```

Program 21

```
RUN

FREQUENCY DIAGRAM

2        4     ★★★★
3        6     ★★★★★★
4        6     ★★★★★★
5        8     ★★★★★★★★
6        19    ★★★★★★★★★★★★★★★★★★★
7        21    ★★★★★★★★★★★★★★★★★★★★★
8        10    ★★★★★★★★★★
9        11    ★★★★★★★★★★★
10       8     ★★★★★★★★
11       4     ★★★★
12       3     ★★★
```

[K] Program 21.

SAQ 5

```
10  REM★★PLURALS★★
20  LET A$="S"
29  PRINT "ENTER WORD"
30  INPUT B$
31  PRINT B$
40  PRINT B$+A$
50  STOP
```

Program 22

Exercise 4

```
10  REM★★SIMPLE CODE★★
20  LET C$=""
30  DIM A$(26)                                ]
40  LET A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"       ]—set up the directory
100 PRINT "NEXT WORD FOR CODING"
101 INPUT W$
102 PRINT W$
110 LET L=LEN(W$)
120 FOR J=1 TO L                   ]   compare each letter of the
130 LET I+1                        ]   word (W$) with each letter in
140 IF W$(J)=A$(I) THEN GOTO 200   ]—— the directory
150 LET I=I+1                      ]
160 GOTO 140                       ]   until found,
200 LET C$=C$+STR$(I)+" " ————————————— then add the index in string
210 NEXT J                             form to the code-string
```

```
220 PRINT
225 PRINT C$
230 STOP
```

Program 23

```
RUN
NEXT WORD FOR CODING? COMPUTING

 3  15  13  16  21  20  9  14  7

RUN
NEXT WORD FOR CODING? PARLIAMENT

 16  1  18  12  9  1  13  5  14  20

RUN
NEXT WORD FOR CODING? PROFESSIONALS

 16  18  15  6  5  19  19  9  15  14  1  12  19
```

[K] **Program 23.**

Exercise 5

```
10  REM★★RANDOM 3-LETTER WORDS★★
20  LET A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
80  PRINT "ANOTHER LIST? ";
81  INPUT R$
82  PRINT R$
90  IF R$<>"YES" THEN GOTO 190
99  REM★★20 WORDS★★
100 FOR K=1 TO 20
109 REM★★WORD STRING EMPTY TO START★★
110 LET W$=""
119 REM★★START OF A WORD★★
120 FOR J=1 TO 3
130 LET X=INT(26★RND+1)
140 LET W$=W$+A$(X)
150 NEXT J
159 REM★★PRINT THE WORD★★
160 PRINT W$
169 REM★★GO BACK FOR NEXT WORD★★
170 NEXT K
180 GOTO 80
190 STOP
```

Program 24

Too many REMs spoil a program's appearance; do they add to its intelligibility?

```
PDY
XNM
HPI
```

EVW
EFT
KZT
IGM
KWL
QHW
AIY
ZBQ
MSH
FFA
ZYJ
BBX
JPT
FSV
BPG
GNR
EEP

How many times must you run to get a 'proper' word?

K Program 24.

UNIT 7
Handling numbers

7.1	Introduction	170
7.2	Averages and the arithmetic mean	170
7.3	Range	173
7.4	Number crunching	176
7.5	Dry running	179
7.6	The representation of numbers	181
7.7	The INT-function for rounding	182
7.8	The ABS-function	184
7.9	Iteration	184
	Assignment 7	189
	Objectives of Unit 7	189
	Answers to SAQ's and Exercises	190

7.1 Introduction

We have demonstrated that computers are not just number crunchers but in this Unit we take a closer look at their arithmetic capacity. We concentrate on fairly straight-forward arithmetic, so don't worry about finding it difficult. We're sure that you will be able to cope. We shall continue the 'let's see what happens if approach', and 'let's get the machine to tell us what is going on'. We hope that you will adopt the same approach with the machine that you use.

7.2 Averages and the arithmetic mean

'How long have you been taking over each Unit of the course, so far?' 'Well, on average, about 3 hours.' If we asked you how long it took you to complete a regular journey, such as going to work, you would answer in a similar manner. Sports enthusiasts use the terms goal average, and batting average, atlases abound with pictures of average rainfalls for the months of the year. We talk of average mark in a test or average age of a group of people, etc.

If we wished to calculate the average or arithmetic mean of the ages of a particular group, we would add up the ages for all of the members of the group and divide that sum by the total number of people in the group. To find the arithmetic mean then involves: adding up, counting and then dividing the sum by the count.

Example 1
Find the arithmetic mean of the following set of numbers:

6,7,2,5,4,4,9,8.

Solution
Their sum = 6+7+2+5+4+4+9+8 = 45.
There are 8 numbers.
So their arithemetic mean = 45/8 = 5.625.

SAQ 1
Find the arithmetic mean of the following set of numbers:

8,4,2,6,1,7,6,1,4.

Arithmetic mean
Example 2
Devise an algorithm and write a program to find the arithmetic mean of the following numbers:

56,47,52,65,24,34,59,37,49,66,
38,24,62,76,31,47,66,61,74,45,
66,44,55,67,36,56,54,54,50,43,
18,83,23,79,29,−9999

Solution
We will express the algorithm first of all in descriptive form.

1. Start.
2. Set counter to 1.

3. Set sum to 0.
4. Input the next mark.
5. If mark = −9999 then go to 9 otherwise carry on to 6.
6. Add mark to sum.
7. Add 1 to counter.
8. Go to 4.
9. Set total to counter−1.
10. Calculate average=sum/total.
11. Output average.

Figure 1 Descriptive algorithm for arithmetic mean

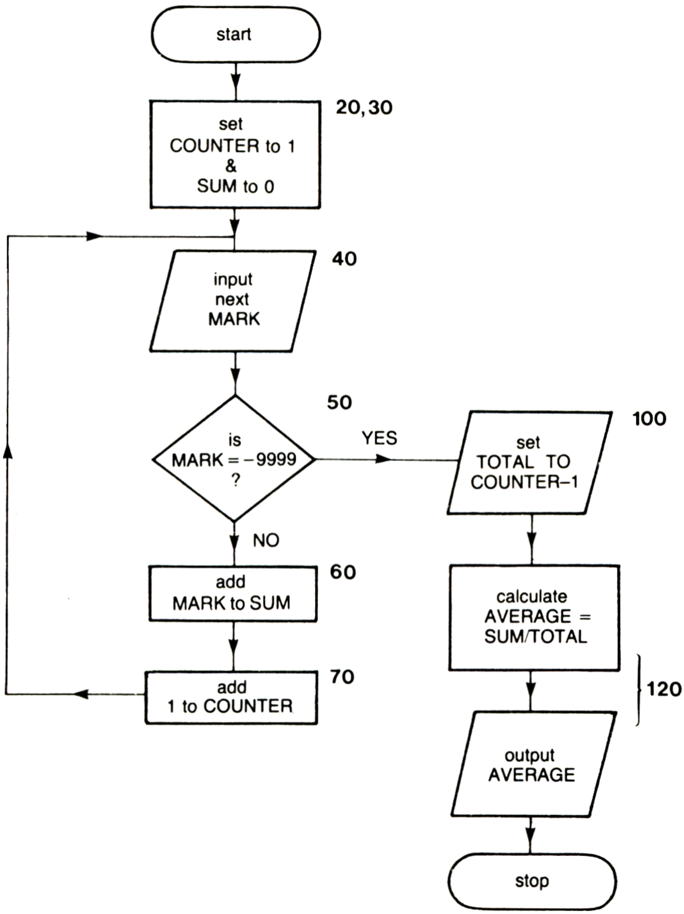


Figure 2 Flowchart for arithmetic mean

```
10  REM★★ARITHMETIC MEAN★★
15  LET C=0
17  LET S=0
20  CLS
30  PRINT "ENTER NEXT NUMBER: END LIST WITH −9999"
40  INPUT N
50  IF N=−9999 THEN GOTO 100
60  LET C=C+1
70  LET S=S+N
80  GOTO 20
100 LET A=S/C
110 CLS
120 PRINT"AVERAGE=";A
130 STOP
```

Program 1   Arithmetic mean

```
RUN
AVERAGE= 50.571428
```

[K] Program 1.

Exercise 1
Write a program to find the average length of the words in the 'Jude' DATA statements of Example 2 of Unit 5. You can do this by grafting a routine onto Program 1 of Unit 5 which already finds the lengths of words.

Exercise 2
Write a program to find the average score in a simulated experiment of tossing a die 100 times.

Simulation
The expected average score when throwing a die a large number of times is:

$$\frac{1+2+3+4+5+6}{6} = \frac{21}{6} = 3.5$$

However, if you look at the answer to Exercise 2, you will see that our run only hit exactly 3.5 once with values ranging from 3.24 to 3.76. That was from the results of 3,000 (30 × 100) throws so you might well ask, 'Is the random number generator biased?' (We did call it 'pseudo' anyway!) How many experiments do we need to convince us that it is, or is not, biased?

To explore that question fully we need to go into statistical theory that is beyond the scope of this course but we can at least find the mean of these means. All we have to do is to take data from the 30 runs of the program:
3.56,3.47,3.52,3.65,. . .
and enter them into Program 1 above. This gives us the mean of the means.

You will notice below that we have entered only the decimal parts of the numbers in order to make our data entry a little easier, e.g. 56 instead of 3.56. (We can do this because all the numbers are 3. something.)

```
56,47,52,65,24,34,59,37,49,66,
38,24,62,76,31,47,66,61,74,45,
66,44,55,67,36,56,54,54,50,43,
-9999

RUN
AVERAGE= 51.266667
```

Figure 3

Thus the overall mean of 3,000 throws is 3.51 to 2 decimal places – a bit more convincing!

Simulation summarised
Simulation is rather a grand word for what we have just done. However, we wanted to emphasize that we can simulate a real life activity without getting deeply involved in statistics. We couldn't toss a die 3,000 times in classroom but with a computer we can collect and process data fairly rapidly.

If your curiosity has been aroused then try the following exercise.

Exercise 3
Write a program to find the average score in a simulated experiment of tossing 2 dice 100 times.

What would you expect the average score to be in this case and in experiments with 3, 4 . . . dice? Are your expectations justified by your experiments?

7.3 Range

While discussing the results of Exercise 2 on page 178, we quite naturally used the idea of range. We said that the values ranged from 3.24 to 3.76. The process involves finding the lowest and highest values of the set.

Example 3
Devise an algorithm and write a routine to find the maximum and minimum values of the numbers stored in the DATA statements in Example 2. Add this routine to the program to find the arithmetic mean as written as a solution to Example 2.

(If you feel confident enough, treat this Example as an exercise before working through our solution.)

Solution
You may feel we've been here before in Unit 4 when we found the lowest member of a list. We could use that approach again but to do so we would first have to put the data in a list form and then sort it twice with the interchange routine. That's a lot of work so we will look at a shorter approach: trying to find the lowest and highest marks as the data is read in.

We know how to input the data (lines 10–50 in Program 1) but what do we do with it as each item is read?

- First we create two stores:
  M for top mark so far
  B for bottom mark so far.

- Then we input the first mark and put it into both B and M. After all it's the lowest and highest so far!
- Then we input each mark and if it is higher than M, put it into M or, if lower than B, put it into B. If neither, just input the next mark.

So a descriptive solution is:

Description

1. Start routine.
2. If counter=1 then write mark into top and bottom and go to 6 otherwise carry on to 3.
3. If mark > top then write mark into top and go to 6 otherwise carry on to 4.
4. If mark >=bottom then got to 6 otherwise carry on to 5.
5. Write mark into bottom.
6. End routine.

Figure 4

And a flowchart:

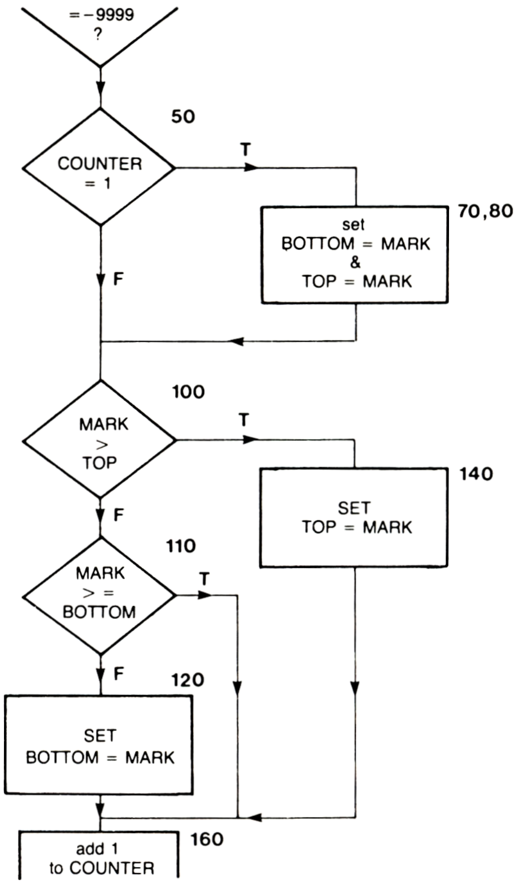


Figure 5.

Which do you find easiest to follow?

Where there are several branches in a program, the descriptive algorithm can be rather confusing. The two dimensional display of the flowchart may be more helpful. It's a matter of personal choice; you judge for yourself!

```
10  REM★★MAX AND MIN★★
20  LET C=1
25  LET S=0
30  PRINT "NEXT NUMBER"
31  INPUT M
32  PRINT M
40  IF M=-9999 THEN GOTO 190
50  IF C>1 THEN GOTO 100
60  REM★★★★★★★★★★★★★★★★
70  LET B=M
80  LET T=M
90  REM★★★★★★★★★★★★★★★★★
100 IF M>T THEN GOTO 140
110 IF M>=B THEN GOTO 160
120 LET B=M
130 GOTO 160
140 LET T=M
150 REM★★★★★★★★★★★★★★★★
160 LET C=C+1
165 LET S=S+M
170 GOTO 30
180 REM★★★★★★★★★★★★★★★★
190 PRINT"MAX=";T,"MIN=";B
200 REM★★★★g★★★★★★★★★★★★
210 LET N=C-1
220 PRINT"AVERAGE=";S/N
230 REM★★★★★★★★★★★★★★★★
```

the program visits this backwater only the first time round the loop when C=1.

the decisions are made here

Program 2

```
RUN
MAX= 83             MIN= 18
AVERAGE = 50.571428
```

[K] Program 2.

Exercise 4
Write a program to draw up a frequency table for the data in Program 2, using categories:

0–9, 10–19, 20–29,. . .   90–99

Suggestion
You could use a score-list

S(0), S(10), S(20),. . . . S(100)

and for each mark read in, test whether it is less than the top of the second band (10), less than the top of the third band (20), etc. until you find

MARK < K true

then increment S(K−10). This is the approach we have used (see answer).

7.4 Number crunching

We have avoided anything other than fairly simple arithmetic so far in the course, and will continue to do so. But it would be wrong not to give a brief insight into the computer's arithmetic capacity. If your heart sinks at the sight of the following few pages, you will miss no vital programming information if you pass on to the next section on dry-running and tracing, but we hope you will give it a try. Our machine certainly does take the drudgery out of arithmetic.

Here is a simple program which calculates for the numbers 1 to 10 their squares (line 50), cubes (line 60) and reciprocals (line 70) and then tabulates the result.

```
  1  REM....TABULATE THE SQUARES, CUBES AND
  2  REM....RECIPROCALS FOR THE FIRST TEN
  3  REM....NATURAL NUMBERS.
 10  PRINT"N";TAB(5);"N★N";TAB(12);"N★N★N";TAB(22);"1/N"
 20  PRINT
 30  FOR I=1 TO 10
 40  LET N=I
 50  LET S=I★I
 60  LET C=I★I★I
 70  LET R=1/I
 80  PRINT N;TAB(5);S;TAB(12);C;TAB(22);R
 90  NEXT I
100  STOP
```

Program 3

```
RUN
N         N★N       N★N★N     1/N

1         1         1         1
2         4         8         .5
3         9         27        .33333333
4         16        64        .25
5         25        125       .2
6         36        216       .16666667
7         49        343       .14285714
8         64        512       .125
9         81        729       .11111111
10        100       1000      .1
```

[K] Program 3.

Raising to a power

We expect that you are familiar with the notation:

$4 \times 4 = 4^2$ (4-squared or 4 raised to the power 2)
$7 \times 7 \times 7 = 7^3$ (7-cubed or 7 raised to the power 3)
$10 \times 10 \times 10 \times 10 \times 10 = 10^5$ (10 raised to the power 5)

In BASIC raised to the power is shown as ↑ (or simply ∧) and ★★ on the ZX81. Any number N raised to the power P is shown as N ↑ P. P is called the exponent and, would you believe it, raising to a power is called exponentiation.

Similarly for negative powers:

	N	P	N ↑ P
$\frac{1}{4}\times\frac{1}{4}=\frac{1}{4\times 4}=4^{-2}$	4	−2	4 ↑ (−2)
$\frac{1}{7}\times\frac{1}{7}\times\frac{1}{7}=\frac{1}{7\times 7\times 7}=7^{-3}$	7	−3	7 ↑ (−3)
$\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}=\frac{1}{10\times10\times10\times10\times10}=10^{-5}$	10	−5	10 ↑ (−5)

Fractional powers (positive and negative) are also possible but we will not be concerned with them.

We can use the ↑ notation instead of ★ in Program 3. Thus Program 3 re-written with ↑ becomes:

```
10   PRINT"N","N★★2","N★★3","N★★(−1)"
20   FOR N=1 TO 10
30   PRINT N,N★★2,N★★3,N★★(−1)
40   NEXT N
50   STOP
```

Program 4

Sequences and their sums

Calculating the individual terms in a sequence, or the sum of the first N terms, is a very great labour without a computer. How long would it take you to evaluate the terms of

$$\frac{1}{1^2}, \frac{1}{2^2}, \frac{1}{3^2}, \ldots \frac{1}{N^2}?$$

Well, it's very easy with Program 5.

```
10   PRINT"N","N★★(−2)"
20   FOR N=1 TO 10
30   PRINT N,N★★(−2)
40   NEXT N
50   STOP
```

Program 5

N	N★★(−2)
1	1
2	.25
3	.11111111
4	.0625
5	.04
6	.027777778
7	.020408163
8	.015625
9	.012345679
10	.01

$\boxed{K}$ Program 5.

Exercise 5
Write a program to find how many terms of the series

$$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$$

are needed to make their sum exceed 2.4.

Exercise 6
Modify the program from Exercise 5 to find out how many terms are needed for

$$\text{sum} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \ldots$$

to exceed 1.5.

Exercise 7
Factorials are interesting numbers. Factorial 4 = 4×3×2×1 and is usually written 4!
So

Factorial 7 = 7×6×5×4×3×2×1 = 7!

Factorial N = N×(N−1)×(N−2)×. . .×1 = N!

Write a program to evaluate the factorial of any positive integer.

Exercise 8
We wrote a rather clumsy program to evaluate the yield (Y) on a deposit (D) at compound interest percentage (P) way back in Unit 1. A neater formula is $Y = D \times (1 + P/100)^T$

where T is the number of years of the investment.

Write a program to evaluate the yield, using this formula, for various deposits, percentages and time periods.

7.5 Dry running

Often we find that a program either does not work at all, or not to our complete satisfaction. If we have reasonable access to a computer we may sit at the machine until we trace the fault, but when all else fails we may be forced to sit down with pencil and paper and think hard. Stepping through an algorithm line by line with pencil and paper is called dry running.

We shall illustrate dry running by looking at Program 18 which was the solution to Exercise 5.

```
10  REM★★SUM OF RECIPROCALS★★
20  LET S=0
30  LET N=1
40  LET S=S+1/N
50  IF S>2.4 THEN GOTO 90
60  LET N=N+1
70  GOTO 40
90  PRINT"SUM=";S;"  THE NO. OF TERMS IS ";N
100 STOP
```

Program 18 (from Exercise 5)

```
RUN
SUM= 2.45   THE NO. OF TERMS IS 6
```

Tracing means finding and recording each step, the line number executed at that step and the values of the variables after that line has been executed. So, for Program 18 we need the headings:

step No.	line No.	N	S
1			
2			
etc			

We omit from the trace lines which don't affect the variables, i.e.:

REM (line 10)
GOTO (line 70)
PRINT (line 90)

Apart from these, the program steps are as follows:

step No.	line No.	N	S
1	20	0	0
2	30	1	0
3	40	1	1

Step No.	line no.	N	S	
4	50	1	1	
5	60	2	1	Note effect of GOTO 40 line 60
6	40	2	1.5	
7	50	2	1.5	
8	60	3	1.5	GOTO again
9	40	3	1.83	
10	50	3	1.83	
11	60	4	1.83	GOTO
12	40	4	2.08	
13	50	4	2.08	
14	60	5	2.08	GOTO
15	40	5	2.28	
16	50	5	2.28	
17	60	6	2.28	GOTO
18	40	6	2.45	

Figure 6

Tracing

Most BASIC interpreters provide a TRACE command, but these often provide so much information that it is difficult to see the wood for the trees. A carefully designed trace routine of your own often works best. We shall not, therefore, show the TRACE command at work but will show you later in this Unit how to write your own trace into a program.

SAQ 2

Complete a dry-run on the following program, starting when line 20 has just been executed and continuing until the condition in line 50 is true. Draw up the table with the same headings as in the above example.

```
10   REM★★SUM OF SQUARES★★
20   LET S=0
30   LET N=1
40   LET S=S+N★★2
50   IF S>50 THEN GOTO 90
60   LET N=N+1
70   GOTO 40
90   PRINT "SUM=";S;"   NO. OF TERMS IS";N
100  STOP
```

Program 6

7.6 The representation of numbers

So far in the course we have left in abeyance a number of questions about the representation of numbers in our programs. We do not intend to consider the mathematics of number representation in computers in general but we need to tidyup our ideas about numbers.

In general terms computers must be able to process and store the following types of number:

(a) positive and negative whole numbers (integers or counting numbers);
(b) fractions and numbers which are partly whole and partly fractional (measuring numbers);
(c) very large and very small numbers;
(d) the number zero.

The single most important piece of advice that we can give you at this stage is that if a number has been involved in any sort of calculation within a program then consider it with a certain amount of suspicion. The reason for this statement is that numbers within a computer are stored and manipulated in binary form, that is to base 2, rather than to the base 10 with which we are familiar. In our familiar decimal notation you will recall that a number like ⅓ or ⅐ is incapable of exact expression, e.g. ⅓ = 0.3333 . . . We assume that by adding on as many 3's to this number as we wish, we can achieve an acceptable level of accuracy in any particular problem. In the binary system, in just the same way, some numbers cannot be expressed exactly, e.g. in decimal form we can say that ⅒ = 0.1 exactly but when this number is changed into binary form it cannot be represented exactly.

Most BASIC interpreters allow for numbers to be expressed to an accuracy of six decimal digits. In decimal form then the number 3⅒ could be represented as 3.10000 to six decimal digit accuracy. However, if this number had been the result of some calculation within a computer, we might find that the number output was 3.09999 or 3.10001. So, in general, you must always be suspicious of the least significant digit in any answer (i.e. the digit on the far right of the number).

Program 7 shows how this type of inaccuracy may occur. The FOR. . . NEXT loop adds 4.0, 4.1 and 4.25 into locations S, T and U one thousand times. Now 4.0 and 4.25 are exactly represented in binary form, but 4.1 is not. We can see that the result of this repetitious summation is exact in respect of 4.0 and 4.25, but not for 4.1 where the error is 0.02 in 4100.00. Obviously, we would avoid writing programs involving such repetitions wherever possible, but we hope that the program will act as a warning about possible inaccuracies.

```
10   REM★★ACCURACY DEMONSTRATION★★
20   LET S=0
30   LET T= 0
40   LET U=0
50   FOR I=1 TO 1000
60   LET S=S+4.0
70   LET T=T+4.1
80   LET U=U+4.25
90   NEXT I
100  PRINT S;TAB(8);T;TAB(21);U
120  STOP
```

Program 7

RUN
4000 4100.02 4250

[K] Program 7.

Small and large numbers
The six to ten decimal digits in microcomputers do not allow us to cope with very small or very large numbers. So these numbers are represented in BASIC in exponential form.

Small numbers

0.000586321 means $\frac{586321}{1,000,000,000} = \frac{586321}{10^9}$

which could be expressed as 586321×10^{-9}. We could also express 0.000586321 as $\frac{0.586321}{1000} = 0.586321 \times 10^{-3}$

In BASIC this last number would be written as 0.586321E−3 or 0.586321E−03.

Similarly,

$0.0234539 = 2.34539 \times 10^{-2} =$ 2.34539E−2

and

$0.00000000959734 = 0.959734 \times 10^{-8} =$ 0.959734E−8.

E stands for exponent, and the base for exponentiation is 10. So E−4 means move the decimal point 4 places to the left, and E+9 means move the decimal point 9 places to the right.

Large numbers

$12368500 = 1.23685 \times 10^7 =$ 1.23685E+7
$935.432 = 0.935432 \times 10^3 =$ 0.935432E+3
959734000000000000000 = 0.959734E+21

Most BASIC interpreters have a range of at least E−32 to E+32; but you should check the details of the particular system which you are using.

[K] Find out how your computer represents small and large numbers with this program.

```
10  REM★★NUMBER DEMONSTRATION★★
20  PRINT"NUMBER","REPRESENTATION"
30  FOR I=−10 TO 10
40  PRINT "10★★";I,10★★I
50  NEXT I
```

Program 8

7.7 The INT-function for rounding

For some problems we need to round off the result of a calculation to the nearest whole number,

e.g. 6.6 to the nearest whole number is 7
7.4 to the nearest whole number is 7.

This is especially true if we are not confident about the last figure accuracy of a decimal number,

e.g. 6.99999 or 7.00001 for 7.

The function INT(X+0.5) does this rounding for us as the following program demonstrates.

```
10  REM★★INT FOR ROUNDING★★
20  PRINT"X";TAB(10);"INT(X)";TAB(20);"INT(X+0.5)"
30  FOR I=-1 4 TO -2.6 STEP(-.1)
40  PRINT I;TAB(10);INT(I);TAB(20);INT(I+0.5)
50  NEXT I
60  STOP
```

X	INT(X)	INT(X+0.5)
−1.4	−2	−1
−1.5	−2	−1
−1.6	−2	−2
−1.7	−2	−2
−1.8	−2	−2
−1.9	−2	−2
−2	−2	−2
−2.1	−3	−2
−2.2	−3	−2
−2.3	−3	−2
−2.4	−3	−2
−2.5	−3	−2
−2.6	−3	−3

Program 9a

Changing line 30 to 30 FOR I=1.4 TO 2.6 STEP(.1) gives

X	INT(X)	INT(X+0.5)
1.4	1	1
1.5	1	2
1.6	1	2
1.7	1	2
1.8	1	2
1.9	1	2
2	2	2
2.1	2	2
2.2	2	2
2.3	2	2
2.4	2	2
2.5	2	3
2.6	2	3

Program 9b

K Program 9a.

SAQ 3
What print-outs will we get from the above program if we change line 30 as follows:

(a) 30 FOR I=.4 TO 2.2 STEP(.2) ?
(b) 30 FOR I=.4 TO −.6 STEP(−.2) ?

7.8 The ABS-function

An arithmetic function related to the above ideas, is to find the modulus, or absolute value of a number. It sounds rather grand, but is very simple.

ABS(X) simply gives us the positive value of X.
e.g. ABS(23)=23, ABS(−23)=23.

The following program illustrates the function.

```
10  REM★★THE ABS-FUNCTION★★
20  PRINT"X";TAB(7);"Y";TAB(14);"X+Y";TAB(21);"ABS(X+Y)"
25  PRINT
30  FOR I=1 TO 4
40  INPUT X
45  INPUT Y
50  PRINT X;TAB(7);Y;TAB(14);X+Y;TAB(21);ABS(X+Y)
60  NEXT I
```

Program 10

RUN

X	Y	X+Y	ABS(X+Y)
5	7	12	12
5	−7	−2	2
−5	7	2	2
−5	−7	−12	12

[K] Program 10.

SAQ 4
What will the print-out table be if we input 9,14,11,−2,−4,13,−7,−8 ?

7.9 Iteration

The process of making a guess at a value, testing it, making a better guess and testing again, etc., until we home in on an item or value is known as iteration. The essence of iteration involves:

- making an arbitrary start
- guessing how accurate this point is
- refining this in a sequence of repeated operations.

Square root by iteration
BASIC can do square roots directly as this program demonstrates:

```
10  FOR X=33 TO 63 STEP (10)
20  PRINT X;TAB(7);X★★(.5);TAB(20);SQR(X)
30  NEXT X
```

Program 11

(SQR(X) gives the square root of X provided that X> = 0.)

```
RUN
33        5.7445627        5.7445627
43        6.5574385        6.5574385
53        7.2801099        7.2801099
63        7.9372539        7.9372539
```

However, we shall also show you how to find a square root by an iterative method not because that's how you would normally do it but because it's an easy example through which to demonstrate iteration.

The method
If you want to square root a number N, then:

- take a guess at the square root, say G
- work out N/G
- then the average of G and N/G is an even better guess than your first one, i.e. it is closer to the unknown square root than your first guess.
- go back to the beginning using this new 'better' guess.

We shan't prove this here (it's in plenty of maths textbooks) but we shall show its working in a simple case.

To square root	$N = 12$
Guess	$G = 2$
Work out	$N/G = 6$
Take average	$\frac{G + N/G}{2} = \frac{2 + 6}{2} = 4$

So 4 is the new guess:

Guess $G = 4$

$N/G = 3$

$\frac{G + N/G}{2} = 3.5$

So 3.5 is the new guess.

In table form this looks like:

G	N/G	$\frac{G+N/G}{2}$
2	12/2=6	(2+6)/2=4
4	12/4=3	(4+3)/2=3.5
3.5	12/3.5=. . .	. . . .

Figure 7

SAQ 5
To make sure that you have grasped the process draw up a table similar to that above, but make your first guess 1.

Iteration. . . stop

The question now is: 'How do we stop the process?' Well, we want to stop the process when the square of our guess becomes as close as possible to N. What do we mean by 'as close as possible'? The answer is that it's up to you. You're in charge! How accurate do you want the square root to be? If, for example, we wish to find the square root of 12 to 2 decimal places, then the difference between G★G and N would have to be

<0.005

We don't need to know whether G★G is bigger or smaller than N – only the difference matters. So we are back to ABS. If

ABS(N−G★G) < 0.005

then stop is what we are after.

Descriptive algorithm for square root iteration

1. Start.
2. Input the number whose square root is sought.
3. Input the accuracy required.
4. Input a guess at the square root.
5. If the guess is within the accuracy required then go to 8 otherwise continue with the next statement.
6. Make the guess more accurate.
7. Return to 5.
8. Output the square root value found.
9. Stop.

Figure 8

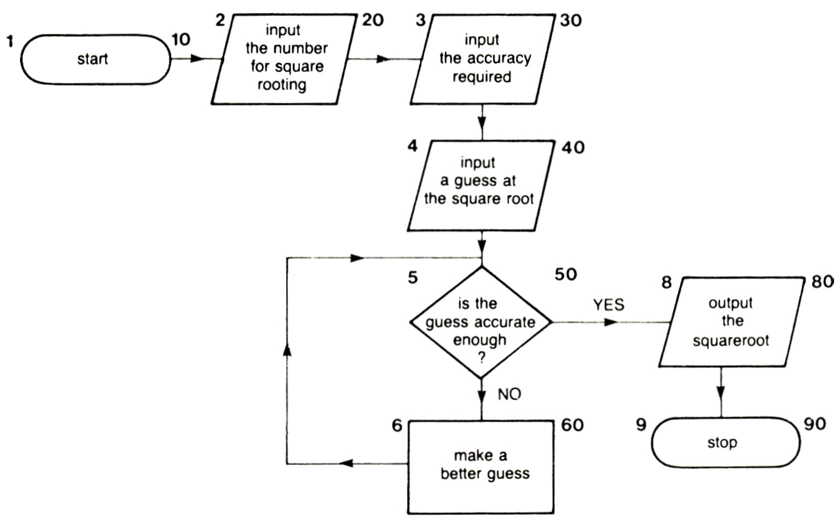


Figure 9 Flowchart for square root iteration

```
10  REM★★SQRT BY ITERATION★★
20  PRINT "NO. FOR SQUARE-ROOTING? ";
21  INPUT N
22  PRINT N
30  PRINT "ACCURACY REQUIRED? ";
31  INPUT A
32  PRINT A
40  PRINT "YOUR GUESS? ";
41  INPUT G
42  PRINT G
50  IF ABS(N-G★G)<A THEN GOTO 80
60  LET G=.5★(G+(N/G))
70  GOTO 50
80  PRINT
81  PRINT"THE SQRT OF ";N;" IS ";G
90  STOP
```

Program 12   Square root iteration

```
RUN
NO. FOR SQUARE-ROOTING? 12
ACCURACY REQUIRED? .005
YOUR GUESS? 2

THE SQRT OF 12 IS 3.4642857

RUN
NO. FOR SQUARE-ROOTING? 12
ACCURACY REQUIRED? .00005
YOUR GUESS? 2

THE SQRT OF 12 IS 3.4641016
```

[K] Program 12.

Tracing the iterative process

Now the result of program 12 is not very startling. As we pointed out earlier on, we can find N directly with a computer. But we wanted a simple example to demonstrate iteration at work so we will now take a closer look at what is happening. We will put a trace into Program 12 as follows:

```
46  PRINT
47  PRINT "OLD";TAB(10);"NEW";TAB(20);"ABS(N-G★G)"
55  PRINT G;
65  PRINT TAB(10);G;TAB(20);ABS(N-G★G)
```

Each pass round the loop (i.e. each iteration) lines 55 and 65 print out a report on how the calculation is going.

```
10  REM★★SQUARE ROOT BY ITERATION★★
20  PRINT "NO. FOR SQUARE ROOTING? ";
21  INPUT N
22  PRINT N
30  PRINT "ACCURACY REQUIRED? ";
31  INPUT A
```

```
32  PRINT A
40  PRINT "YOUR GUESS? ";
41  INPUT G
42  PRINT G
46  PRINT
47  PRINT"OLD";TAB(10);"NEW";TAB(20);"ABS(N-G★G)"
50  IF ABS(N-G★G)<A THEN GOTO 80
55  PRINT G;
60  LET G=.5★(G+(N/G))
65  PRINT TAB(10);G;TAB(20);ABS(N-G★G)
70  GOTO 50
80  PRINT
81  PRINT "THE SQRT OF ";N;" IS ";G
90  STOP
```

Program 13   Iterative square root with trace

```
RUN
NO. FOR SQUARE-ROOTING? 12
ACCURACY REQUIRED? .005
YOUR GUESS? 2
OLD          NEW          ABS(N-G★G)
 2            4            4
 4            3.5          0.24999999             3 loops only
 3.5          3.4642857     .00125755059

THE SQRT OF 12 IS 3.4642857
```

```
RUN
NO. FOR SQUARE-ROOTING? 12
ACCURACY REQUIRED? .0005 ——— increasing the accuracy tenfold still only
YOUR GUESS? 2                takes 4 loops
OLD          NEW          ABS(N-G★G)
 2            4            4
 4            3.5          0.24999999
 3.5          3.4642857     .0012755059
 3.4642857    3.4641016    2.6077032E-8

THE SQRT OF 12 IS 3.4641016
```

K Program 13.

```
RUN
NO. FOR SQUARE-ROOTING? -67 ———— if we give it a negative number
ACCURACY REQUIRED? .005
YOUR GUESS? 8
OLD          NEW          ABS(N-G★G)
8            -0.1875       67.035156
-0.1875      178.57292     31955.287
178.57292    89.09886      8005.6068
```

89.09886	44.173443	2018.2931
44.173443	21.328347	521.8984
21.328347	9.0934941	149.69163
9.0934941	0.86279454	67.744414
0.86279454	−38.395923	1541.2469
−38.395923	−18.325473	402.82296
−18.325473	−7.33468	120.79753
−7.33468	0.90000312	67.81006
0.90000312	−36.772092	1419.1867
−36.772092	−17.475029	372.37663
−17.475029	−6.8204933	113.51913
−6.8204933	1.5014216	69.254267
1.5014216	−21.561476	531.89726
−21.561476	−9.2270412	152.13829
−9.2270412	−0.98288766	67.966068
−0.928288766	33.5918	1195.409
33.5918	15.798633	316.5968
15.798633	5.7788798	

it doesn't like it, and we have to pull out the plug!

Exercise 9

Change the square root program above to produce cube roots. If G is a guess at the cube root of N then ½(G + N/G^2) will be a better guess.

Assignment 7

1. If g is a guess at the solution of the quadratic equation $x^2+bx+c = 0$, then a better guess is $-c/(b+g)$. Devise an algorithm and write a BASIC program to find the solution of any quadratic equation.

   Notes:

   1. By solution we mean a value of x which makes the left-hand side of the equation equal to zero.
   2. When testing your algorithm choose values of b and c such that $b^2>4c$.

2. Pythagoras' theorem was about right-angled triangles, remember. It does however generate some special whole number (integer) triples, e.g.

   $3^2 + 4^2 = 5^2$

   $5^2 + 12^2 = 13^2$

   Devise an algorithm and BASIC program to find out how many such integer triples there are for numbers <100.

   Hint: you will need to use the INT(X+0.5) function in this program.

Objectives of Unit 7

Calculate (manually) an arithmetic mean. ☐

Write a program to calculate arithmetic means. ☐

Write a program to find the largest and smallest item in a data list. ☐

Use ★, / and ★★ in programs. ☐

Dry run a program. ☐

Interpret numbers in E notation. ☐

Use INT(X+.5) for rounding. ☐

Use ABS(X). ☐

Write programs for iterative routines to include terminating procedures. ☐

Insert trace print lines in a program. ☐

Answers to SAQ's and Exercises

SAQ 1

Sum = 8 + 4 + 2 + 6 + 1 + 7 + 6 + 1 + 4 = 39
There are 9 numbers.
So the arithmetic mean is 39/9 = 4.333. . .

Exercise 1

```
10  REM★★AVERAGE LENGTH★★
20  LET S=0
30  LET C=1
40  PRINT "NEXT WORD"
42  INPUT W$
44  PRINT W$
50  IF W$="ZZZZ" THEN GOTO 160
60  LET L=LEN(W$)
70  LET S=S+L
80  LET C=C+1
90  GOTO 40
150 ★★★★★★★★★★★★★★★★★★★★
160 LET N=C-1
170 LET A=S/N
180 PRINT"AVERAGE LENGTH OF THE WORDS IS ";A
190 STOP
```

Program 14

[K] Program 14.

Exercise 2

```
10  REM★★MEAN OF 100 THROWS★★
20  REM★★ OF 1 DIE★★
50  LET S=0
60  FOR I=1 TO 100
70  LET X=INT(6★RND+1)
80  LET S=S+X
90  NEXT I
```

```
100 PRINT"AVERAGE SCORE=";S/100
110 STOP
```

Program 15

[K] Program 15.

Exercise 3

```
10  REM★★MEAN OF 100 THROWS★★
20  REM★★OF 2 DICE★★
50  LET S=0
60  FOR I=1 TO 100
70  LET X=INT(6★RND+1)
75  LET Y=INT(6★RND+1)
80  LET S=S+X+Y
90  NEXT I
100 PRINT"AVERAGE SCORE=";S/100
100 STOP
```

Program 16

[K] Program 16.

Exercise 4

```
10  PRINT ". . . . . . HISTOGRAM . . . . . ."
20  DIM S(101)
40  REM
45  PRINT
46  PRINT "DATA . . . (RANGE 0–109)"
50  INPUT M
51  IF M=−9999 THEN GOTO 50
52  IF M<0 OR M>109 THEN GOTO 147
55  PRINT M;" ";
80  LET K=11
90  IF M<K THEN GOTO 120
100 LET K=K+10
110 GOTO 90
120 LET S(K−10)=S(K−10)+1
130 GOTO 50
145 PRINT
147 PRINT "TABLE . . ."
148 PRINT
150 FOR K=1 TO 101 STEP (10)
160 PRINT K−1;"–";TAB(7);S(K);TAB(12);
165 FOR P=1 TO S(K)
166 PRINT "▒";
167 NEXT P
168 PRINT
170 NEXT K
180 STOP
```

This bit prints a histogram as well.
Try it on your screen.

Program 17

```
RUN
 0        0
 10       1
 20       4
```

30	5
40	6
50	8
60	7
70	3
80	1
90	0
100	0

[K] Program 17.

Exercise 5

```
10  REM★★SUM OF RECIPROCALS★★
20  LET S=0
30  LET N=1
40  LET S=S+1/N
50  IF S>2.4 THEN GOTO 90
60  LET N=N+1
70  GOTO 40
90  PRINT"SUM=";S;"   THE NO. OF TERMS IS   ";N
100 STOP
```

Program 18

```
RUN
SUM= 2.45   THE NO. OF TERMS IS 6
```

[K] Program 18.

Exercise 6

```
10  REM★★SUM OF N★★(−2)★★
20  LET S=0
30  LET N=1
40  LET S=S+N★★(−2)
50  IF S>1.5 THEN GOTO 90
60  LET N=N+1
70  GOTO 40
90  PRINT "SUM=";S,"NO. OF TERMS=";N
100 STOP
```

Program 19

```
RUN
SUM= 1.5117971   NO. OF TERMS= 7
```

You can, of course, change line 50 to explore the number of terms needed for the sum to exceed other values, e.g.

```
50  IF S>1.6 THEN GOTO 90
RUN
SUM= 1.6004969   NO. OF TERMS= 22 ———— for 1.6

50  IF S>1.61 THEN GOTO 90
RUN
SUM= 1.611039   NO. OF TERMS= 29 ———— for 1.61
```

```
50  IF S>1.62 THEN 90
RUN
SUM= 1.620244  NO. OF TERMS= 40  ———————— for 1.62

50  IF S>1.63 THEN 90
RUN
SUM= 1.6301195  NO. OF TERMS= 67  ———————— for 1.63
```

What a good starting point these and similar programs would make to lessons about limits and convergence of number series!

K Program 19.

Exercise 7

```
10  REM. . . .FACTORIALS
20  PRINT"TO END THE RUN, ENTER -9999"
30  PRINT
40  PRINT "NEXT FACTORIAL? ";
41  INPUT N
42  PRINT N
50  IF N=-9999 THEN GOTO 130
60  LET F=1
70  FOR I=1 TO N
80  LET F=F★I
90  NEXT I
100 PRINT N,F
110 PRINT
120 GOTO 40
130 STOP
```

Program 20

```
RUN
TO END THE RUN, ENTER -9999

NEXT FACTORIAL? 1
 1            1

NEXT FACTORIAL? 3
 3            6

NEXT FACTORIAL? 5
 5            120

NEXT FACTORIAL? 7
 7            5040

NEXT FACTORIAL? 9
 9            362880

NEXT FACTORIAL? 11
 11           39916800
```

```
NEXT FACTORIAL? -9999

>READY
```

[K] Program 20.

Exercise 8

```
10  REM★★COMPOUND INTEREST★★
20  PRINT"ENTER DEPOSIT,PERCENTAGE,TIME"
30  INPUT D
31  PRINT D;", ";
34  INPUT P
35  PRINT P;", ";
37  INPUT T
38  PRINT T
39  PRINT
40  PRINT"TIME","YIELD"
50  FOR I=1 TO T
60  PRINT I,D★(1+P/100)★★I
70  NEXT I
80  STOP
```

Program 21

```
ENTER DEPOSIT,PERCENTAGE,TIME
 500 , 11.5 , 5
 TIME             YIELD
 1                557.5
 2                621.6125
 3                693.09794
 4                772.8042
 5                861.67669


ENTER DEPOSIT, PERCENTAGE, YIELD
1000, 13.5, 7
TIME              YIELD
1                 1135
2                 1288.225
3                 1462.1354
4                 1659.5237
5                 1883.5594
6                 2137.8399
7                 2426.4482
```

[K] Program 21.

SAQ 3

(a)

X	INT(X)	INT(X+0.5)
.4	0	0
.6	0	1
.8	0	1
1	1	1
1.2	1	1
1.4	1	1
1.6	1	2
1.8	1	2
2	2	2

(b)

X	INT(X)	INT(X+0.5)
.2	0	0
0	0	0
−.2	0	0
−.4	−1	0
−.6	−1	0
	−1	−1

SAQ 4

RUN

X	Y	X+Y	ABS(X+Y)
9	14	23	23
11	−2	9	9
−4	13	9	9
−7	−8	−15	15

SAQ 5

9	N/G	$\frac{G+N/G}{2}$
1	12	6.5
6.5	1.8	4.15
4.15	2.89	3.52
3.52	3.41	3.46 etc

Exercise 9

```
10  REM★★CUBEROOT ITERATION★★
20  PRINT "NUMBER FOR CUBEROOTING? ";
21  INPUT N
22  PRINT N
30  PRINT "ACCURACY REQUIRED? ";
31  INPUT A
32  PRINT A
40  LET G=N/2 ——————— the machine makes the first guess at N/2
```

```
43  LET C=1
50  IF ABS(N-G★G★G) <A THEN GOTO 80
60  LET G=0.5★G+(N)/G★G))
67  LET C=C+1
70  GOTO 50
80  PRINT"THE NO. OF LOOPS=";C
81  PRINT"THE CUBEROOT OF  ";N;"=";G
90  END
```

Program 22

```
RUN
NUMBER FOR CUBEROOTING? 28
ACCURACY REQUIRED? .005
THE NO. OF LOOPS= 14
THE CUBEROOT OF 28 = 3.0364096

RUN
NUMBER FOR CUBEROOTING? 10101
ACCURACY REQUIRED? .005
THE NO. OF LOOPS= 28
THE CUBEROOT OF 10101 = 21.616634

RUN
NUMBER FOR CUBEROOTING? -937
ACCURACY REQUIRED? .005
THE NO. OF LOOPS= 21
THE CUBEROOT OF -937 =-9.7854198
```

[K] Program 22.

UNIT 8
Introduction to data processing

8.1	Introduction	198
8.2	Sorting	198
8.3	Subroutines	202
8.4	Searching	206
8.5	Tables	212
	Assignment 8	218
	Objectives of Unit 8	219
	Answers to SAQ's and Exercises	219

8.1 Introduction

In this Unit we shall emphasize again how important it is to impose some sort of order on data. In particular, we shall analyse in detail one method of ordering data: the interchange procedure for sorting. Having sorted the data we shall then show how to search through it quickly using the bisection search procedure. Finally we shall look at how to handle data in tabular form.

These activities will also give us a chance to see how subroutines can help us perform many of those little repetitive tasks which can occur in programs of any size.

8.2 Sorting

In Unit 4 we spent some time discussing the procedure for finding the lowest value item in a list. We did this by an interchange procedure that put the lowest item into position 1 on the list. We said that we could repeat the procedure for the rest of the list, placing the second lowest value into position 2, etc., and we left you with the problem of sorting the whole list as an assignment. Because of the interchange sort's importance, we are now going to look at it in greater detail.

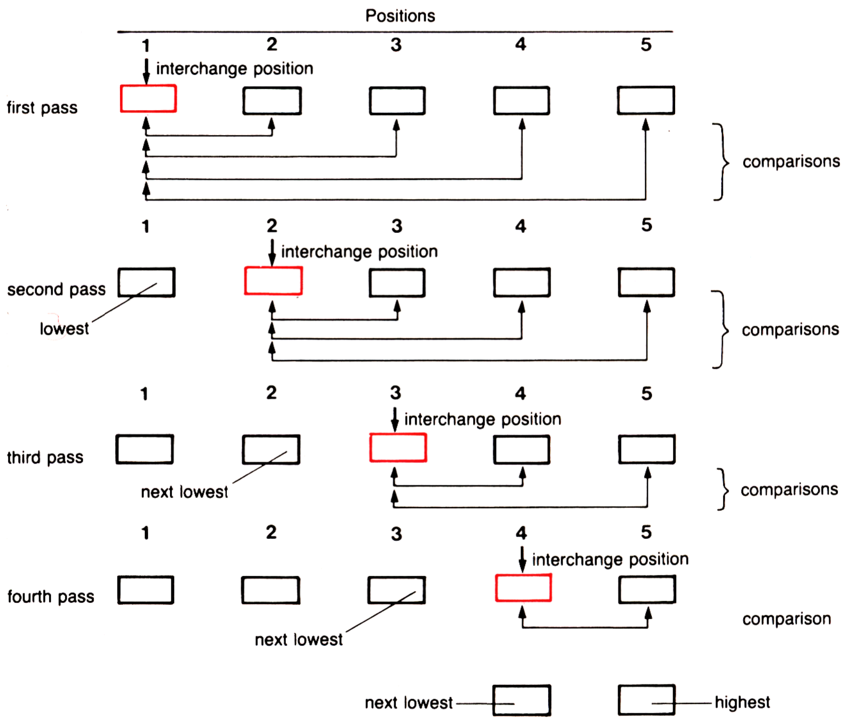


Figure 1 The sort procedure for a list of 5 items

Figure 1 illustrates the procedure for placing the items into locations 1 to 5 with the lowest item in 1, the next lowest in 2 and so on.

First pass. On the first pass all items are compared with the item in position 1 and the lowest is then placed in position 1.

Second pass. Position 1 can now be ignored and the procedure repeated on positions 2 to 5. This will find the next lowest which is placed in position 2.

Third pass. Now positions 1 and 2 can be ignored since they contain 'lowest' and 'next to lowest'. The procedure is repeated on positions 3 to 5. This finds the third lowest which goes in position 3.

Fourth pass. This is performed on items 4 and 5 only and results in the 4th lowest going into the fourth position. The remaining item must be the highest and will already be in the fifth position so no further passes are needed.

We can summarise the sort procedure as:

loop number	point interchange	remaining sub-sequence	
		start	end
1	1	2	5
2	2	3	5
3	3	4	5
4	4	5	5

Figure 2a Four sorts in a list of 5 items

Or, more generally, if we want to sort a list of N items:

loop number	interchange point	remaining sub-sequence	
		start	end
1 2 3 . . . K . . . N−1	1 2 3 . . . K . . . N−1	2 3 4 . . . K+1 . . . N	N . . . N . . . N N N

Figure 2(b) (N−1) sorts in a list of N items

Since each pass involves a repetitive series of comparisons, it is an obvious candidate for a FOR. . . NEXT. . . loop. Then we need a further loop to decide which loop we are going round:

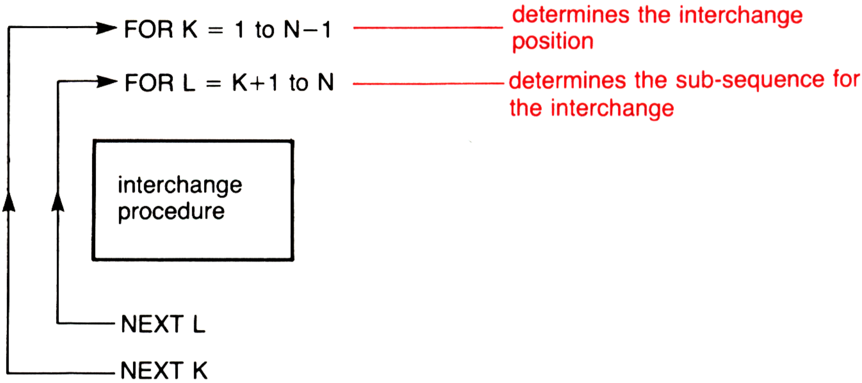


Figure 3 The nested loops of interchange sorting

SAQ 1
Use the interchange sort to place the following in order. Show the numbers stored at each location after each run

6, 1, 4, 0, 2, 3, 7, 8

The program is:

outer loop decides interchange point

if the item in the sub-sequence is >= the item in the interchange position then do not interchange

```
210  REM★★SORT ROUTINE★★
220  FOR K=1 TO N−1
230  FOR L=K+1 TO N
240  IF X$(L) >= X$(K) THEN GOTO 280
250  LET T$=X$(L)
260  LET X$(L)=X$(K)
270  LET X$(K)=T$
280  NEXT L
290  NEXT K
300  REM★★END OF SORT ROUTINE★★
```

the 'power-house': should be condensed onto one line to speed up process

inner loop decides the sub-sequence

Program·1 Interchange sort

Using the sort program
The sort program can be used whenever it is needed. Here is one particular use: to sort a list of names into alphabetical order.

lines 50–100 read in the data
lines 210–300 carry out the sort
lines 410–450 print out the sorted list
the data has been stored in line 900

```
10  PRINT "........SORT ROUTINE........"
11  PRINT
15  REM★★DATA MAX LENGTH=6★★
30  DIM X$(100,6)
50  LET I=1
60  PRINT "NEXT DATA? ";
70  INPUT X$(I)                                   reading in the data
71  PRINT X$(I)
80  IF X$(I)="ZZZZ" THEN GOTO 190
90  LET I=I+1
100 GOTO 60
180 REM★★★★★★★★★★★★★★★★★★★★
185 REM★★LENGTH OF LIST★★
190 LET N=I−1
200 REM★★★★★★★★★★★★★★★★★★★★
210 REM★★SORT ROUTINE★★★★★★
220 FOR K=1 TO N−1
230 FOR L=K+1 TO N
240 IF X$(L) >= X$(K) THEN GOTO 280
250 LET T$=X$(L)                                  sort routine
260 LET X$(L)=X$(K)
270 LET X$(K)=T$
280 NEXT L
290 NEXT K
300 REM★★END OF SORT ROUTINE★★
400 REM★★★★★★★★★★★★★★★★★★★★
410 PRINT"FINAL SORTED LIST"
420 FOR P=1 TO N
430 PRINT X$(P);" ";                              printing out the result
440 NEXT P
450 PRINT
500 REM★★★★★★★★★★★★★★★★★★
```

Program 2 Using the sort routine

```
RUN
NEXT DATA? TONY
NEXT DATA? SAM
NEXT DATA? PETE
NEXT DATA? JOE
NEXT DATA? BILL
NEXT DATA? ZZZZ
FINAL SORTED LIST
BILL   JOE   PETE   SAM   TONY
```

8.3 Subroutines

By the time you have reached this stage you will begin to distinguish the wood from the trees. You will be aware that programs have an overall structure and are assemblies of smaller parts like the paragraphs of an essay. It is usual to break a program down into its constituent parts, and to write and test each part separately. Certain operations are often repeated several times throughout a program. The structure of a program may be simplified and tidied up by including these repetitive operations as subroutines.

We shall illustrate subroutines by taking a final look at the sort procedure. We are going to insert two extra trace print lines into the program so that we can see what is happening at each of the three stages of the sort routine:

Sort routine	Trace print line to show
1. Input.	The list as taken in.
2. Sort.	The list after each sub-sequence.
3. Output.	The final, sorted, list.

Figure 5 shows the overall structure and how we can use one PRINT subroutine for all three PRINT operations.

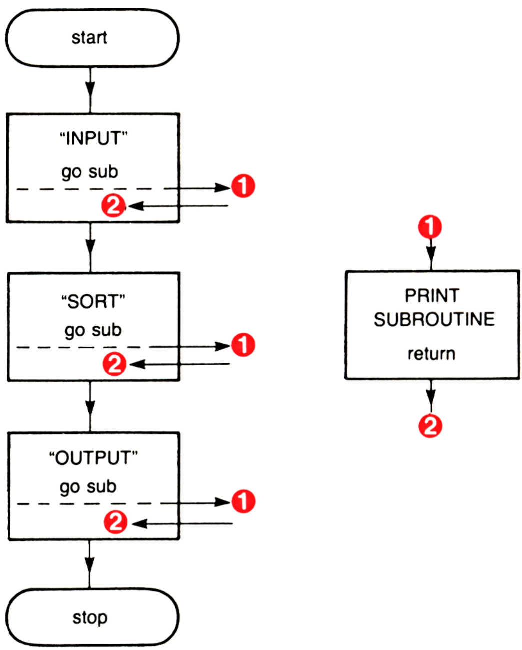


Figure 5 Print sub-routine in the sort program

GOSUB

In BASIC to go to a subroutine we say:

GOSUB

followed by the line number of the start of the subroutine. Each subroutine must end with the statement

RETURN

which will return control to the next line in the main body of the program after the appropriate GOSUB statement. Thus in the following program segment line 30 transfers control to line 200 and lines 200 and 210 are executed. The 220 returns control to line 40 for the program to continue in the normal way.

```
10   INPUT A
20   INPUT B
30   GOSUB 200
40   LET S=A+B
50   . . . . .

200  PRINT "A","B"
210  PRINT A,B          GOSUB 200
220  RETURN
```

SAQ 2
What is the value of B after this program has been run: (a) if 5 is inputted; (b) if 3 is input?

```
10  INPUT A
20  IF A<5 THEN GOTO 40
30  GOSUB 70
40  LET B=A★A
50  PRINT B
60  STOP
70  LET A=1/A
80  RETURN
```

Program 3

Here is the sort program with a print subroutine (lines 500 – 550) which is used each time the program executes line 194, line 280 and line 420.

```
10   PRINT ". . . . . . . . SORT ROUTINE . . . . . . . ."
11   PRINT
15   REM★★DATA MAX LENGTH=6★★
30   DIM X$(100,6)
50   LET I=1
60   PRINT "NEXT DATA? ";
70   INPUT X$(I)
71   PRINT X$(I)
80   IF X$(I)="ZZZZ" THEN GOTO 190
90   LET I=I+1
100  GOTO 60
180  REM
185  REM★★LENGTH OF LIST★★
190  LET N=I−1
```

```
191 SCROLL                                  Moves print up screen
192 PRINT "LIST AT START"                   when screen is full
194 GOSUB 510                               First trace
196 PRINT
200 REM
210 REM★★SORT ROUTINE★★
220 FOR K=1 TO N-1
225 PRINT"PASS NO.";K
230 FOR L=K+1 TO N
240 IF X$(L) >= X$(K) THEN GOTO 280
250 LET T$=X$(L)
260 LET X$(L)=X$(K)
270 LET X$(K)=T$
280 GOSUB 510                               Second trace
285 NEXT L
287 PRINT
290 NEXT K
300 REM★★END OF SORT ROUTINE★★
400 REM
405 SCROLL
410 PRINT "FINAL SORTED LIST"
420 GOSUB 510                               Output
450 STOP
500 REM★★PRINT SUBROUTINE★★
510 FOR P=1 TO N
515 SCROLL
520 PRINT X$(P);" ";                        Subroutine
530 NEXT P
540 SCROLL
550 RETURN
```

Program 4 Print subroutine in sort program

```
RUN
LIST AT START
TONY     SAM     PETE     JOE     BILL      printed by GOSUB at
                                            line 194

PASS NO. 1
SAM      TONY    PETE     JOE     BILL
PETE     TONY    SAM      JOE     BILL
JOE      TONY    SAM      PETE    BILL
BILL     TONY    SAM      PETE    JOE

PASS NO. 2
BILL     SAM     TONY     PETE    JOE       each block printed
BILL     PETE    TONY     SAM     JOE       by GOSUB at line
BILL     JOE     TONY     SAM     PETE      280 on the four occa-
                                            sions the program
PASS NO. 3                                  executes the loop
BILL     JOE     SAM      TONY    PETE      controlled by K
BILL     JOE     PETE     TONY    SAM
```

```
PASS NO. 4
BILL           JOE       PETE      SAM       TONY

FINAL SORTED LIST
BILL           JOE       PETE      SAM       TONY
```

printed by GOSUB at line 420

Examples on subroutines
The purpose of a subroutine is to simplify and shorten long programs. By its very nature then, it is difficult to get short meaningful programs which illustrate subroutines without their often being a little contrived. We need a program where the same or similar function is repeated at different points in the program.

Example 1
The game of dice ('craps' in the USA) provides a simple example. A pair of dice is thrown twice and the total score on each throw is noted. If the two scores are the same, the game ends. If they are different, the dice are thrown again. Write a program to simulate the game which prints out the number of throws required to obtain equal scores and what that score was.

Solution

```
10  PRINT ". . . . . . . . 2 EQUAL THROWS . . . . . . . ."
15  PRINT
30  LET C=1
39  REM★★1ST THROW★★
40  GOSUB 130
50  LET S1=S
59  REM★★2ND THROW★★
60  GOSUB 130
70  LET S2=S
80  IF S1=S2 THEN GOTO 100
90  LET C=C+1
95  GOTO 40
100 PRINT "EQUAL SCORE ";S1;" IN ";C;" THROWS"
110 STOP
120 REM★★DICE ROLLING SUBROUTINE★★
130 LET D1=INT(6★RND+1)
135 LET D2=INT(6★RND+1)
140 LET S=D1+D2
150 RETURN
```

The subroutine produces generally different values of S for the 'main' program.

subroutine

Program 5 Simulation of 'craps'

```
RUN
EQUAL SCORE 7 IN 9 THROWS
RUN
EQUAL SCORE 8 IN 21 THROWS
RUN
EQUAL SCORE 8 IN 7 THROWS
RUN
EQUAL SCORE 5 IN 4 THROWS
RUN
```

EQUAL SCORE 10 IN 22 THROWS
RUN
EQUAL SCORE 10 IN 7 THROWS

Exercise 1

(a) Write a segment of program to print a line of 32 dashes "– – – – – – – –" across the screen.

(b) Write one line of program to print a 'submarine' or <=>, at any point across the screen or printer where the variable S determines the position.

(c) Write a program to print on successive lines:
  (i) a line of dashes;
  (ii) a submarine at any point;
  (iii) another line of dashes;
  with the line printing in (i) and (iii) in a subroutine.

Exercise 2

Now you must admit that the solution to Example 1 looks like a vessel in a canal, so why not a submarine as we are concerned with subroutines? Instead of battleships in a 2-dimensional sea, we have a submarine in a 1-dimensional canal. Anyway, we have the picture for a simple game.

Write a program to generate a random number between 1 and 30. The submarine is going to take up the last three positions of the width (30, 31, 32). Use the random number to print the submarine in random positions along the canal.

Exercise 3

The essence of the game we are going to play with the machine will be clear from Exercise 2. The computer generates a random number and invites you to find the submarine by guessing a number between 1 and 32. If you guess the correct position, i.e. between S and S+2 if S is the random number (remember the submarine takes up 3 places in the line), then the machine records a 'hit' and the game ends. If you don't find the submarine, the machine will record a 'miss' and invite you to try again. Write a program to do this.

(We advise you to write the program to give yourself the option to stop playing before you find the submarine, because it is infuriating to have to try every position across the screen just to stop the program running. You could just pull the plug out, and then it would sink!?)

8.4 Searching

The submarine problem gives us a good lead into discussions about searching data. The only methodical way to find the submarine was to search the canal successively position by position starting from one end. How much easier it would have been had the program responded with 'too high' or 'too low', as appropriate, after each guess. No doubt you can immediately think of a procedure for 'homing-in' on the submarine as quickly as possible!

Similarly, if dictionaries, telephone directories, encyclopedias and library cata-

logues were not arranged in alphabetical order, think how difficult it would be to find the desired information.

But if we've gone to a lot of trouble to sort our **data** into numerical or alphabetical order, then we need an efficient search technique to find any given item. If we consult a dictionary or telephone directory for an item, we don't start looking at the first page and work methodically through the volume page by page until the item is found. We take a rough guess, e.g. if the name begins with 'P' then we try to open the directory at just over half-way through it, and start looking from that point.

Bisection search

A 'rough' guess is too imprecise a term for a computer. However we can specify guessing points as follows:

- divide the range of items into half and ask 'is the item above or below the half-way mark?'
- if it is below then we define a new range with the middle item now acting as the upper limit;
- if above then the middle item becomes our lower limit;
- either way we discard half the old range and repeat our halving or bisection procedure with the new range.

So the bisection search is, in outline:

Is 7 in the list 1,2,3,4,5,6,7,8,9,10?

List in order:

1	2	3	4	5	6	7	8	9	10

Halve list.

1 2 3 4 5	6 7 8 9 10

Is 7 = middle item? No.
Is 7 < middle item? No.
So 7 is in top half.

Halve list.

6 7 8	8 9 10

Is 7 = middle item? No.
Is 7 < middle item? Yes.

Halve list.

6 7	7 8

Is 7 = middle item? Yes.
So 7 is in list.

That outline illustrates the principle of the bisection search but in practice we need to distinguish between the values of the items in a list and the indexes of those items.

Example 2

An ordered list contains the items A, F, I, M, P, T, U, Z. Use the bisection search procedure to find whether or not P is in the list.

We call P, Query – the value we wish to enquire about:

Index	1	2	3	4	5	6	7	8
Item	A	F	I	M	P	T	U	Z
Start-Index	Low (1)							High (8)
Mid-Index, $\text{Int}\frac{(1+8)}{2}=4$				Mid (4)				

Comparisons

is Query = Item (4)? no!
is Query < Item (4)? no!
make Index (4) the new Low

Index	4	5	6	7	8
Item	M	P	T	U	Z
Start-Index	Low (4)				High (8)
Mid-Index, $\text{Int}\frac{(4+8)}{2} = 6$			Mid(6)		

Comparisons

is Query = Item (6)? no!
is Query < Item (6)? yes!
make Index(6) the new High

Index	4	5	6
Item	M	P	T
Start-Index	Low(4)		High(6)
Mid-Index, $\text{Int}\frac{(4+6)}{2} = 5$		Mid(5)	

Comparison

is Query = Index (5)? yes!

Figure 6 Bisection search

Exercise 4
Carry out the bisection search procedure on the list in Example 2 but looking for the letter I.

We only had to make 3 comparisons to home-in on the item 'P' in Example 2, but they were a bit long-winded, and the whole procedure may seem to have little advantage over simply searching straight through the list. The effectiveness of the method is not really apparent in short lists. We will demonstrate its power in searching longer lists later, but first we still have some loose ends to tie up.

Some problems with bisection search

(a) How to stop

Example 3
Carry out the same procedure as before, but search for the letter 'Q'.

The method would proceed exactly the same as before as far as the 3rd comparison, so we'll pick up the story there.

Query = Q

Index	4	5	6
Item	M	P	T
Start-Index	Low(4)	⋮	High(6)
Mid-Index, $\text{Int}\left(\frac{4+6}{2}\right) = 5$		Mid(5)	

Comparisons

is Query = Item(5)? no!
is Query < Item(5)? no!
make Index(5) the new Low

Index	4	5	6
Item		P	T
Start-Index		Low(5)	High(6)
Mid-Index, $\text{Int}\left(\frac{5+6}{2}\right) = 5$		Mid(5)	

Comparisons

is Query = Item(5)? no!
is Query < . . . . we have done this before?!

So we don't seem able to stop. Q is not there but we are stuck looking for it between P and T. We have already met the problem of stopping the process in the last example. If the indexes Low and High have moved so close that they are at adjacent positions, and Query is not yet found, then Query is not a member of the list. That is the end and outcome of the search. So the end is either when the Query has been found, or when Low and High occupy adjacent indexes (High – Low = 1).

(b) How to start

To start the process seemed straightforward enough. We make Index (1)=Low and Index(N)=High. Trouble would occur however if Item(1) and Item(N) were not the lowest and highest possible values.

E.g. consider the following list which does not include letters before C or after S:

1	2	3	4	5
C	F	G	P	S
Low				High

If Query was A or B or higher than S, then the process would not work. The easiest solution is to ensure that the items at the ends of the list will always have the extreme values, e.g. in a list of names make Item(1)=AAAA and Item(N)=ZZZZ.

We will now outline the algorithm in flowchart form.

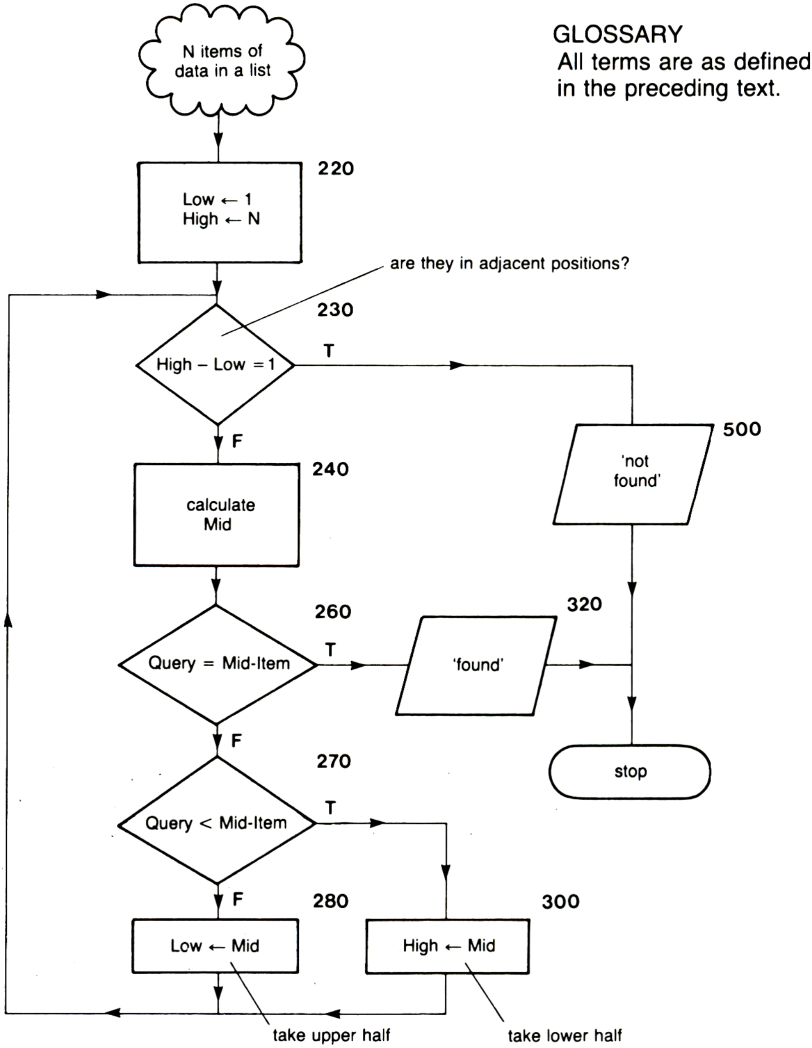


Figure 7 Flowchart for bisection search

All we have to do now is to write the program:

```
10  REM★★BISECTION SEARCH★★
20  DIM N$(13,7)
25  DIM Q$(7)
30  DIM T$(13,4)
40  LET N$(1)="AAAA"
41  LET N$(2)="BENNY"
42  LET N$(3)="COPPER"
```

```
43  LET N$(4)="DRAPER"
44  LET N$(5)="EDDIE"
46  LET N$(6)="GWYNNE"
47  LET N$(7)="HETTY"
48  LET N$(8)="MORLEY"
49  LET N$(9)="PROSSER"
50  LET N$(10)="SMYTHE"
51  LET N$(11)="WEEKS"
52  LET N$(12)="WILSON"
53  LET N$(13)="ZZZZ"
60  LET T$(1)="0000"
61  LET T$(2)="1234"
62  LET T$(3)="9832"
63  LET T$(4)="1980"
64  LET T$(5)="7294"
65  LET T$(6)="5821"
66  LET T$(7)="8632"
67  LET T$(8)="7832"
68  LET T$(9)="1383"
69  LET T$(10)="1147"
70  LET T$(11)="5529"
71  LET T$(12)="9936"
72  LET T$(13)="9999"
90  REM
95  LET N=13
100 REM★WE ARE USING ZZZZ THIS TIME★
150 PRINT "QUERY NAME? ";
151 INPUT Q$
152 PRINT Q$
200 REM★★START OF SEARCH★★
205 PRINT
210 PRINT "L";TAB(5);"H";TAB(10);"M";TAB(15);"N$(M)"
220 LET L=1
225 LET H=N
230 IF H-L=1 THEN GOTO 500
240 LET M=INT((L+H)/2)
250 PRINT L;TAB(5);H;TAB(10);M;TAB(15);N$(M)
260 IF Q$=N$(M) THEN GOTO 320
270 IF Q$<N$(M) THEN GOTO 300
280 L=M
290 GOTO 320
300 LET H=M
310 GOTO 230
320 REM★★END OF SEARCH★★
330 PRINT Q$;"'S TELE NO. IS  ";T$(M)
350 GOTO 110
500 PRINT Q$;"  IS NOT IN THE LIST"
550 GOTO 110
```

Program 6   Bisection search program

```
RUN
QUERY NAME? MORLEY
 L    H     M     N$(M)
 1    13    7     HETTY
 7    13    10    SMYTHE
 7    10    8     MORLEY
MORLEY'S TELE NO. IS 7832
DO YOU WISH TO LOOK FOR ANOTHER NAME?
QUERY NAME? WEEK
 L    H    M    N$(M)
 1    13   7    HETTY
 7    13   10   SMYTHE
 10   13   11   WEEKS
WEEK IS NOT IN THE LIST
```

8.5 Tables

When we want to store a lot of information there are various methods open to us. One is lists (see Unit 4), which are sometimes called one-dimensional arrays. A second method is tables or two-dimensional arrays.

Suppose you want to store the following data:

	1st qtr	2nd qtr	3rd qtr	4th qtr
Car sales	20	70	80	40
Servicing	10	14	18	11
Petrol	30	45	50	30

Figure 8 Income for Main Road Service Station (£,000's)

Now you could put this in one list but it would be hard to use. The first four items would be income for car sales, the next four for servicing, etc. Alternatively you would have three lists: one for car sales, one for servicing and one for petrol. But ZX81 BASIC allows you to have a two dimensional table named by any of the 26 letters of the alphabet, e.g.:

T(,)

Comparison of lists and tables
Lists need one index to describe a position in the list. Tables need two, which are usually called sub-scripts not indices (or indexes, as we have called them).

List
L(1), L(2), L(3) . . .   L(I) . . .

index of this item = 3

Array

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

this item needs two sub-scripts:
3 to tell us it is in row 3;
2 to tell us it is in column 2.

Tables

- A table must contain either all string variables, or all numerical variables. (Numbers can of course be stored as strings, and their values found by the VAL-function.)
- We use one of the 26 letters to describe the table as a whole, e.g. A table, B$ table, M$ table.

Generally, a table comprises:

	col.1	col.2	col.3	col.4
row 1	r1c1	r1c2	r1c3	....
row 2	r2c1	r2c2	r2c3	....
row 3	r3c1	r3c2	r3c3	....
etc	....	....	....	....

Figure 9 The rows and columns of a table

For the service station data, T needs 3 rows and four columns and so contains 12 items:

T(1,1)	T(1,2)	T(1,3)	T(1,4)
T(2,1)	T(2,2)	T(2,3)	T(2,4)
T(3,1)	T(3,2)	T(3,3)	T(3,4)

So

T(2,1) = 10

T(3,3) = 50 etc.

This is very similar to the idea of tables which you have previously met. There we said that a file consists of a series of records each of which consists of fields. In table form this would look like:

	Field 1	Field 2
	Name	Telephone number
Record 1	BENNY	1234
Record 2	COPPER	9823
Record 3	DRAPER	1850
Record 4	EDDIE	7294

Or, more generally:

	Field 1	Field 2	Field 3	etc
Record 1	R1F1	R1F2	R1F3	. . . .
Record 2	R2F1	R2F2	R2F3	. . . .
Record 3	R3F1	R3F2	R3F3	. . . .
etc	. . . .	. . . .	. . . .	. . . .

If the telephone numbers table is called T$ then the individual items will be labelled:

	Field 1	Field 2
	Name	Telephone number
Record 1	T$(1,1)=BENNY	T$(1,2)=1234
Record 2	T$(2,1)=COPPER	T$(2,2)=9823
Record 3	T$(3,1)=DRAPER	T$(3,2)=1950
Record 4	T$(4,1)=EDDIE	T$(4,2)=7294

- The whole table is called T$ table.
- Each item in the table is described by two subscripts. Thus 1950 (3rd row, 2nd column) is

  T$(3,2)
- The 3 and 2 describe the position of item T$(3,2), not its value. Its value is 1950.

So we say

T$(3,2) = 1950

In general

T$(R,C)

Row subscript — Column subscript

Example 4
The N$-table overleaf has 9 values as shown. What are their variable names?

	R \ C	1	2	3
	1	BENNY	COPPER	DRAPER
N$(,)	2	EDDIE	GWYNNE	HETTY
	3	MORLEY	PROSSER	SMYTHE

Solution

BENNY = N$(1,1) COPPER = N$(1,2) DRAPER = N$(1,3)

EDDIE = N$(2,1) GWYNNE = N$(2,2) HETTY = N$(2,3)

MORLEY = N$(3,1) PROSSER = N$(3,2) SMYTHE = N$(3,3)

SAQ 3
In the following A$ table identify the variables and their values as in Example 4.

ARCHER	BENNY	COPPER
DRAPER	EDDIE	FRAME
GWYNNE	HETTY	KEMP
LAMB	MORLEY	NOAKES
PROSSER	SMYTHE	TAIT

DIM for arrays
Numeric arrays If you want a two dimensional numerical array then you must tell the computer what size it will be by using a DIM statement. e.g. to store the table

4 columns

A( )=

	Table for 12 numbers		

3 rows

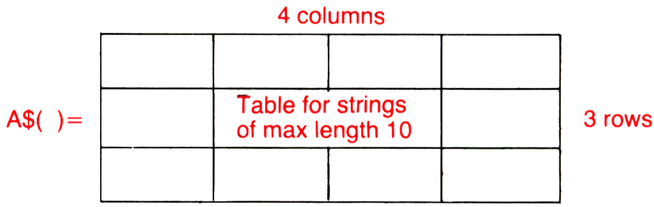
you insert

DIM A(3,4)

rows columns

in the program *before* your first use of A( ).

String arrays The same applies to two dimensional string arrays but, as with string lists you must decide on the maximum string length. So to store the table



you need to say

DIM A$(3,4,10)

max string length
columns
rows

Tables and nested loops

If FOR. . . NEXT loops and lists seemed to be made for each other, then even more so do nested FOR. . . NEXT loops and tables seem complementary.

For example, suppose you want to read:

ARCHER,BENNY,COPPER,DRAPER,EDDIE,FRAME,GWYNNE,HETTY, KEMP,LAMB,MORLEY,NOAKES,PROSSER,SMYTHE,TAIT,WEEKS

into a table, N$, with 4 rows and 4 columns. (We need a string array because we are storing string data.) This can be done with a READ statement in two nested loops:

```
60  FOR I=1 TO 4
90  FOR J=1 TO 4
80  READ N$(I,J)
90  NEXT J
100 NEXT I
```

This process is carried out in full by lines 10 to 100 of Program 7.

It's all very well to store the value in a table, but of course we cannot see the result of this until we print it out. The second half of the program prints the table values out in a column with I and J accompanying them so that you can identify clearly how I and J are used.

```
10  PRINT ". . . . . TABLE INPUT AND PRINT . . . . ."
15  PRINT
20  PRINT "TABLE SET UP FOR 16 INPUTS . . ."
```

```
30  DIM N$(20,5,7)          DIM statement for 2D array
40  REM★★INPUT ROUTINE★★
60  FOR I=1 TO 4
70  FOR J=1 TO 4
80  PRINT "NEXT INPUT? ";   input routine
81  INPUT N$(I,J)
82  PRINT N$(I,J);
83  PRINT TAB(30);4★(I-1)+J
90  NEXT I
100 NEXT J
110 REM★★PRINT ROUTINE★★
111 PRINT
112 PRINT ". . . . . TABLE FULL . . . . ."
113 PAUSE 100
114 POKE 63465,255
115 CLS
116 PRINT "I";TAB(10);"J";TAB(20);"N$(I,J)"
130 FOR I=1 TO 4
140 FOR J=1 TO 4
150 PRINT I;TAB(10);J;TAB(20);N$(I,J)
160 NEXT J
180 NEXT I
200 STOP
```

Program 7 Reading data into a 4 × 4 array.

RUN

I	J	N$(I,J)
1	1	ARCHER
1	2	BENNY
1	3	COPPER
1	4	DRAPER
2	1	EDDIE
2	2	FRAME
2	3	GWYNNE
2	4	HETTY
3	1	KEMP
3	2	LAMB
3	3	MORLEY
3	4	NOAKES
4	1	PROSSER
4	2	SMYTHE
4	3	TAIT
4	4	WEEKS

SAQ 4
The following amendments are made to Program 7. Write out what the output table will look like.

```
60  FOR I=1 TO 3
70  FOR J=1 TO 5
```

```
130 FOR I=1 TO 3
140 FOR J=1 TO 5
```

Table output

The output of Program 7 is not very satisactory since we want to see the table in table form. To do this we delete line 115 and insert a new print routine:

```
130   FOR I=1 TO 5
140   FOR J=1 TO 3
150   PRINT TAB(10★(J−1));N$(I,J);
160   NEXT J
170   PRINT
180   NEXT I
190   GOTO 270
```

the first column starts at position 1−1=0

columns 10 characters wide

Program 8

The output then is:

```
ARCHER      BENNY       COPPER
DRAPER      EDDIE       FRAME
GWYNNE      HETTY       KEMP
LAMB        MORLEY      NOAKES
PROSSER     SMYTHE      TAIT
```

Assignment 8

1. A salesman has 4 product lines. The value (in £) of his firm orders for one week are shown in the table.

day \ product	1	2	3	4	totals
1	500	300	20	25	e
2	600	700	40	0	f
3	200	550	60	20	g
4	250	450	100	5	h
5	400	200	100	11	i
totals	a	b	c	d	t

Write a program which will help him analyse his week's work by giving:

(i) his day totals (e,f,g,h,i)
(ii) his product totals (a,b,c,d)
(iii) his overall weekly total (t).

2. Write a program to extend the submarine game to a 10 × 10 grid. If the guess is close to the submarine then the program should give a 'near miss' clue. You decide what is meant by 'close'.

Objectives of Unit 8

Now that you have completed this Unit, check that you are able to:

- Use the interchange sort (manually) on a set of data ☐
- Write two nested program loops to perform the interchange sort ☐
- Follow GOSUB in programs ☐
- Write GOSUBS into programs ☐
- Use the bisection search (manually) on a set of data ☐
- Write a program for the bisection search ☐
- Put data into two dimensional arrays ☐
- Write a program to read data into a two dimensional array ☐
- Write a program to print data out of a two dimensional array ☐
- Write a program to find the row sums and the column sums in a two dimensional array ☐

Answers to SAQ's and Exercises

After 1st run	0	6	4	1	2	3	7	8
After 2nd run	0	1	6	4	2	3	7	8
After 3rd run	0	1	2	6	4	3	7	8
After 4th run	0	1	2	3	6	4	7	8
After 5th run	0	1	2	3	4	6	7	8
After 6th run	0	1	2	3	4	6	7	8
After 7th run	0	1	2	3	4	6	7	8

SAQ 2

(a) B = 1/25

(b) B = 9 (GOSUB is not used in this case.)

Exercise 1

(a)
```
10  FOR I=1 TO 32
20  PRINT"-";
30  NEXT I
40  PRINT
```
Program 11

(b) PRINT TAB(S);"<=>"

(c)
```
15  INPUT S
16  PRINT S
20  GOSUB 100
30  PRINT TAB(S);"<=>"
40  GOSUB 100
50  STOP
100 FOR I=1 TO 32
110 PRINT"-";
120 NEXT I
130 PRINT
```

```
140 RETURN
```

Program 12

The value that we give S will determine the position of the submarine along the canal, and we get picture like:

```
------------------------------------------------------------
                             <=>
------------------------------------------------------------
```

Exercise 2

```
10   REM★★SUBMARINE★★
50   LET S=INT(29★RND+1)
60   GOSUB 510
70   PRINT TAB(S);"<=>"
80   GOSUB 510
90   STOP
500  REM★★PRINT SUBROUTINE★★
510  FOR I=1 TO 60:PRINT"-";:NEXT I:PRINT
520  PRINT "-";
530  NEXT I
540  PRINT
550  RETURN
RUN
------------------------------------------------------------
                          <=>
------------------------------------------------------------
RUN
------------------------------------------------------------
  <=>
------------------------------------------------------------
RUN
------------------------------------------------------------
                                                 <=>
------------------------------------------------------------
```

Program 13

Exercise 3

```
10   REM★★SUBMARINE★★
30   REM★★PRINT CHALLENGE★★★
40   GOSUB 300                          → 1
50   PRINT "A NUMBER FROM 1 TO 29 MIGHT FIND ME"
60   GOSUB 300                          → 2
70   REM★★RANDOM POSITION OF SUB★★            subroutine used 8 times
80   LET S=INT(29★RND+1)
90   PRINT
100  PRINT "TRY ANOTHER NUMBER? ";
101  INPUT X
```

```
102 PRINT X
110 IF X<S THEN GOTO 190
120 IF X>S+2 THEN GOTO 190
130 REM★★A HIT★★
140 GOSUB 300                          → 3
150 PRINT TAB(S);"HIT"
160 GOSUB 300                          → 4
170 GOTO 320
180 REM★★A MISS★★
190 GOSUB. 300                         → 5
200 PRINT"YOU MISSED"
210 GOSUB 300                          → 6
220 INPUT"DO YOU STILL WANT TO PLAY";R$
230 IF R$="YES" THEN GOTO 90
240 PRINT"SPOILSPORT!! I WAS HERE":PRINT
250 GOSUB 300                          → 7
260 PRINT TAB(S);"<=>"
270 GOSUB 300                          → 8
280 GOTO 320
290 REM★★PRINT SUBROUTINE★★
300 FOR I=1 TO 60                      subroutine prints just one
305 PRINT"-";                          line of dashes
310 NEXT I
311 PRINT
315 RETURN
```

Program 14

```
RUN
------------------------------------------------------------
A NUMBER FROM 1 TO 60 MIGHT FIND ME
------------------------------------------------------------

TRY ANOTHER NUMBER? 27
------------------------------------------------------------
YOU MISSED
------------------------------------------------------------
DO YOU STILL WANT TO PLAY? YES
TRY ANOTHER NUMBER? 60
------------------------------------------------------------
YOU MISSED
------------------------------------------------------------
DO YOU STILL WANT TO PLAY? NO
SPOILSPORT!! I WAS HERE
------------------------------------------------------------
  <=>
------------------------------------------------------------
```

Exercise 4

1	2	3	4	5	6	7	8
A	F	I	M	P	T	U	Z

Mid (4)

Query = Item (4)? No.
Query < Item (4)? Yes.
make Index (4) the new high

A	F	I	M
1	2	3	4

Mid-Index $= \mathrm{Int}\left(\frac{1+4}{2}\right) = 2$

Query = Item (2)? No.
Query < Item (2)? No.
make Index (2) the new low

2	3	4
F	I	M

Query = Item (3)? Yes.
Therefore I is in list

SAQ 3

A$(1,1) = ARCHER
A$(2,1) = DRAPER
A$(3,1) = GWYNNE
A$(4,1) = LAMB
A$(5,1) = PROSSER

A$(1,2) = BENNY
A$(2,2) = EDDIE
A$(3,2) = HETTY
A$(4,2) = MORLEY
A$(5,2) = SMYTHE

A$(1,3) = COPPER
A$(2,3) = FRAME
A$(3,3) = KEMP
A$(4,3) = NOAKES
A$(5,3) = TAIT

SAQ 4

RUN

I	J	N$(I,J)
1	1	ARCHER
1	2	BENNY
1	3	COPPER
1	4	DRAPER
1	5	EDDIE
2	1	FRAME

2	2	GWYNNE
2	3	HETTY
2	4	KEMP
2	5	LAMB
3	1	MORLEY
3	2	NOAKES
3	3	PROSSER
3	4	SMYTHE
3	5	TAIT

(Note that WEEKS was not read into the table. A 5×3 table will only read the first 15 items.)

INDEX

ABS 190
Absolute value **184**
Algorithm **45,** 51, 54, 76, 79, 81, 151, 170, 175, 179, 186, 209
Alphabetical order **76,** 78, 79, 108
American Standard Code for Information Interchange **87**
Arithmetic 28, **170**, 176, 184
  mean **170**
  operators 18, **23**, 25
Array **91–92**
  one dimensional **91**, 212–16
  two dimensional 91, **212–16**
  and FOR . . . NEXT loops 97
ASCII Code **87**
Assignment statement 18, 23
Average **170–1**, 185

Backing store **9**
Base 2 181
  10 181
BASIC **10**, 46, 64, 69, 70, 73, 76, 91, 96, 118, 121, 125, 177, 180, 181, 182, 184, 202, 212
Beginners All-purpose Symbolic Instruction Code **10**
Binary code **76, 87**, 181
Bisection search 198, **207–12**
Blank lines 100
Brackets 91
Branch 40, 42, 44, 47, 175

Central processing unit **9**, 17, 23
Character **64**, 66, 70, 118, 158–9
Checks **134**, 139
Codes 76, 158
Code-breaking 119
Columns 126–28, 148, 213, 216
Comma
  in PRINT statements **37–8**
Commands **14–15**
Comparisons 53, 200, 208
  of strings 76
Concatenation **157**
Copying **21–22**
Counting 40, **48–52**, 93, 97, 119–21, 125, 151, 153–4, 170–1
Cube 176

Data **16–18**, 79, 172, 198
  processing 75
  file 76, 79
  check 134

Decimal numbers 26, 172, 181, 183
Decision box **46–7**
DIM 91–2, 124, 160, 215–16, 217
Dimension 91
Drawing a line 104
Dry running **179**
Dummy value **41–2**, 55

Entry **15**–16
Equal sign **18**
Error message 136
Errors 134
Execution 14, **16**, 18, 23, 39
Exponentiation **177**, 182

Factorial **178**
Faults **179**
Fields **75**, 77, 134, 214
Files **77**, 213
Filing 129
Flowcharts **45–9**, 51, 54, 80, 94, 106, 135, 152, 154, 171, 174, 186, 215
  loops in flowcharts 101–2
FOR . . . NEXT . . . loop **98–102**, 123, 124, 181, 200, 216
  nested **102–4**, 200
  interchange 107
Frequency 119–25
  diagrams **123–5**, 127–8
  table **119–22**, 155
Fractional power **177**
Fractions 187

Games **146**, 147, 158, 205
GOSUB 202–11
GOTO 39, 40, 48, 179
Graphics
  drawing a line 101
  patterns 103–4
Guess **184–5**, 207

Highest so far **173–75**
High-level programming language 10

IF . . . THEN . . . 40–1
Increment 176
Index 91–2, 120–1, 160, 208–9, 212
Inequalities **42–5**
INPUT 12, **17–18**
INPUT ". . ."; . . . 69
Input device **9**, 17
Input list 91–2
INT 148–9, 150, **182–3**
Integer 95, **148,** 178
Interchanging **105–9**, 173, 198
  interchange sort 92, **204–7**
Interpreter **10**, 19, 26

Item 92
Iteration **184–8**

Jump 41, 44
Jumps of convenience 41, 44

Keywords 16

Laws of chance 144
LEFT 130, 134
LEN 118, 132
LET 17–18, 23
Line numbers **13**, 41
LIST 15
List(s) **90**, 105, 173, 198, 212
  counting **93–4**, 122
  input and output 92–6
  numeric lists 91
  one dimensional lists 91
  score lists 155–6
  sorting 198–204
  string lists 92
  variables 90
Literal printing **28–9**, 36–7
LOAD 15
Logical state of the assertion **43**
Logical structure **40–1**
Loop 48–9, 128, 157, 175, 187–8, 200
  counter 151
  in flowcharts 101–2
  FOR . . . NEXT 97–102, 181, 216
  nested 102–4
  nested FOR . . . NEXT **102–4**
  perpetual 40
  and print patterns 103–4
Lowest so far 82, 173–4
Low-level programming language 10

Machine code **10**
Mark 91
Mean **170–3**
  memory location – see Store location
MID$ 130
Modulus **184**

Names
  choosing store location names 19, 20
  common system of location names 21
  sorting 105–9
Negative power **177**
Nested loops **102–4**
  FOR . . . NEXT loops **102–4**, 200, 216
Newline **12**, 14, 18
NEXT 97
Notation **90**, 177
Number line **43**
Numbers 64, 66, 67, 70, 159
  number crunching 176
  large 182
  representation of 181
  small 182
  whole **148–50**, 182–3
Numeric variables 90, 213
Numerical constants **26**
Numerical order 76
Output device **9**, 17, 18, 67
Output display 100
Overwriting **22**

PRINT 17, **18**, 28
  PRINT " " **29**, 36, 37, 66, 69
  PRINT . . ., 36
  PRINT . . .; . . . 66
  commas in **PRINT** statements 37, 38, 66
  subroutine 202
Print layout 71, 103–4, 123
  statement 70
  zones **37**, 38, 66
Problem solving 10–11
Program 10, **11**, 16–17, 45
  programming style **40**
Prompts **28**, 69, 74
Pseudo–random numbers 145, 172

Quotation marks 18, **29**
  with strings **65**

Raising to a power **177**
Random numbers **144–8**, 150–1, 172
RAND 145–6
Range **173**
Reciprocal 176
Records **75**, 77, 79, 214
Relationships **42**
REM 27, 179
Repetition **38–9**, 48–9, 97, 200
  and inaccuracy 181
Response **16**
RIGHT$ **130**
RND 145–7
  RND+1 147–8
Rogue value **41**
Rounding 182–3
Rows **213**, 216
RUN 14, 15, 38

SAVE 15
Self-Assessment Questions **11**
Sequence
  of instructions 10, **11**, 13
  of numbers 144–5
  of random numbers 145–9
  and their sums **177**
Scores 153, **155–6**, 175

Searching **79**, 198, 206–212
Sets 90, 147, 173
Simulation **172–3**
Skip lines 100–1
Sorting 53–4, **79–81**, 105–6, 173, **198–201**
Square 176
Square root **184–8**
Standard letters **73–5**
Statement numbers **12–13**
Statistical analysis 119
  averaging 144
STEP
  with **FOR . . . NEXT** loop 97–100
STOP 11, 14
Store location **18–26**, 82, 90–1, 96, 181, 200
  for strings **64**, 66, 69, 70, 118
  numeric 70
  temporary 105–6
Storage notation 90
Strings **64–70**, 118, 122, 133, 157, 159
  comparing 76
  cutting up **129–30**
  lists 91
  string array 216
  string variables **90**, 213
STR 158–9
Style, programming **40**
Subroutines **202–5**
Sub–scripts **213**, 215
Sub-strings **131–2**
Symbols
  $ **64**
  loops in flowcharts **101–2**

TAB 125–8
  variable 128
Tables 212–18
  output 218
Tabulation 125–28
Tally marks **120**
Terminating value **41**, 55
Trace **179–80**, 187–8
Trace **67–8**, 71, 179–80

Unconditional jump **40**

VAL 133–6, 213
Variable notation **90**, 213
Variable **TAB** 128
Variables 64, 90

Whole numbers 148–9, 182–3
Words 66

Course comments
Please let us know what you think of this course to help us improve future editions of it. We would especially like to know of any problems you have had in getting particular programs to run on your micro.

Make of microcomputer M27X

Name ....

Address ....

....

....

Correspondence tuition

Are you studying by yourself and need help?

If you are, enrol now with NEC as a correspondence student on 30 Hour BASIC. We will give you a correspondence tutor with experience of your make of microcomputer. He will then take you through the course, marking and commenting on your assignments and giving you any advice you need on getting the programs in 30 Hour BASIC to run on your micro.

To enrol, send us the following details:

Name ....

Address ....

....

Postcode ....

Tel. No. ....

Date of birth ....

Microcomputer you are using ....

M27X

Course fee: £30 (£25 if you already have a copy of 30 Hour BASIC).

Also available

Cassettes

Two cassettes of the main programs in 30 Hour BASIC. Price £5.95 inc. p & p and VAT each. (Suitable for use with the BBC Computer only.)

Free notes

30 Hour BASIC needs small adaptations for use on certain microcomputers. If you are having problems getting our programs to work on your machine, send us a stamped addressed envelope with the make and model of your microcomputer written clearly on the inside of the envelope flap. We'll send you a sheet of notes on getting 30 Hour BASIC programs to run on your computer.

National Extension College,
18 Brooklands Avenue, Cambridge CB2 2HN.

FlexiStudy
—NEC—
RECOMMENDED TEXT ®

30 Hour BASIC ZX81 Edition

What is BASIC?

Microcomputers are the tool of the 80's. BASIC is the language that all of them use. So the sooner you learn BASIC, the sooner you will understand the microcomputer revolution.

30 Hour Basic is a simple self-instructional course on the language of microcomputers. But programs need more than language: they need structure as well. So the course also teaches you good programming techniques. You'll learn how to keep, order and sort files, records and directories; how to print letters and addresses; how to invent your own computer games; how to handle numbers and so on.

Can anyone take the course?

Yes. You don't need a microcomputer to do the course: you can study it by yourself or as an NEC correspondence student. But you will get more out of it if you have your own microcomputer or access to one. That way you'll need to allow extra time for keying and running programs.

Which computer?

30 Hour BASIC has been specially prepared for The Computer Programme (BBC TV) but can be used with any microcomputer except the ZX80. This is the ZX81 Edition.

£5.50 ISBN 0 86082 301 6

Cover design: Peter Hall

M27X

30 Hour BASIC ZX81

BBC

NEC

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/